MATLAB工程应用丛书

Simulink
建模与仿真

姚俊 马松辉 编著

西安电子科技大学出版社

http://www.xduph.com

目　录

- 第6章　Simulink系统仿真原理
- 第7章　Simulink子系统技术
- 第8章　Simulink命令行仿真技术
- 第9章　S-函数
- 第10章 控制系统设计分析
- 第11章 DSP Blockset

谢谢使用！

- 策 划：马武装
- 制 作：李林娜
- 监 制：梁家新
- 单 位：西安电子科技大学出版社

- 电 话：029-8228788
- 传 真：029-8232746
- 主 页：http://www.xduph.com
- E-mail：xdupkj@163.com

第1章　　绪　论



1.1 动态系统的计算机仿真

1.1.1 系统与模型

为了能全面、正确地理解系统仿真，需要对系统仿真所研究的对象进行概要的了解。这里对与系统仿真相关的知识——系统与系统模型进行简单的介绍。

1. 系统

系统是指具有某些特定功能，相互联系、相互作用的元素的集合。这里的系统是指广义上的系统，泛指自然界的一切现象与过程。它具有两个基本特征：整体性和相关性。整体性是指系统作为一个整体存在而表现出某项特定的功能，它是不可分割的。

对于任何系统的研究都必须从如下三个方面考虑：

(1) 实体：组成系统的元素、对象。

(2) 属性：实体的特征。

(3) 活动：系统由一个状态到另一个状态的变化过程。

组成系统的实体之间相互作用而引起的实体属性的变化，通常用状态变量来描述。研究系统主要研究系统的动态变化。除了研究系统的实体属性活动外，还需要研究影响系统活动的外部条件，这些外部条件称之为环境。

2. 系统模型

系统模型是对实际系统的一种抽象，是对系统本质(或是系统的某种特性)的一种描述。模型可视为对真实世界中物体或过程的信息进行形式化的结果。模型具有与系统相似的特性，可以以各种形式给出我们所感兴趣的信息。

模型可以分为实体模型和数学模型。实体模型又称为物理效应模型，是根据系统之间的相似性而建立起来的物理模型。实体模型最常见的是比例模型，如风洞吹风实验常用的翼型模型或建筑模型。数学模型包括原始系统数学模型和仿真系统数学模型。原始系统数学模型是对系统的原始数学描述。仿真系统数学模型是一种适合在计算机上演算的模型，主要是指根据计算机的运算特点、仿真方式、计算方法、精度要求将原始系统数学模型转换为计算机程序。

数学模型可以分为许多类型。按照状态变化可分为动态模型和静态模型。用以描述系统状态变化过程的数学模型称为动态模型。而静态模型仅仅反映系统在平衡状态下系统特征值间的关系，这种关系常用代数方程来描述。按照输入和输出的关系可分为确定性模型和随机性模型。若一个系统的输出完全可以用它的输入来表示，则称之为确定性系统。若系统的输出是随机的，即对于给定的输入存在多种可能的输出，则该系统是随机系统。

离散系统是指系统的操作和状态变化仅在离散时刻产生的系统，如交通系统、电话系统、通信网络系统等等，常常用各种概率模型来描述。连续系统模型还可分为集中参数的和分布参数的，线性的和非线性的，时变的和时不变的，时域的和频域的，连续时间的和离散时间的等等。表1.1列出了各种类型的数学模型及其数学描述。

表1.1 数学模型分类

模型类型	静态系统模型	动态系统模型			
		连续系统模型			离散系统模型
		集中参数	分布参数	离散时间	
数学描述	代数方程	微分方程 状态方程 传递函数	偏微分方程	差分方程 离散状态方程	概率分布 排队论

1.1.1计算机仿真

1. 仿真的概念

仿真是以相似性原理、控制论、信息技术及相关领域的有关知识为基础，以计算机和各种专用物理设备为工具，借助系统模型对真实系统进行试验研究的一门综合性技术。它利用物理或数学方法来建立模型，类比模拟现实过程或者建立假想系统，以寻求过程的规律，研究系统的动态特性，从而达到认识和改造实际系统的目的。

系统仿真涉及相似论、控制论、计算机科学、系统工程理论、数值计算、概率论、数理统计、时间序列分析等多种学科。

相似性原理是仿真主要的理论依据。所谓相似，是指各类事务或对象间存在的某些共性。相似性是客观世界的一种普遍现象，它反映了客观世界不同事物之间存在着某些共同的规律。采用相似性技术建立实际系统的相似模型就是仿真的本质过程。

2. 仿真分类

按照实现方式的不同可以将系统仿真分为如下几类：

(1) 实物仿真：又称物理仿真。它是指研制某些实体模型，使之能够重现原系统的各种状态。早期的仿真大多属于这一类。它的优点是直观形象，至今仍然广泛应用。但是为系统构造一套物理模型，将是一件非常复杂的事情，投资巨大，周期长，且很难改变参数，灵活性差。

(2) 数学仿真：数学仿真就是用数学语言去表述一个系统，并编制程序在计算机上对实际系统进行研究的过程。这种数学表述就是数学模型。数学仿真把研究对象的结构特征或者输入输出关系抽象为一种数学描述(微分方程、状态方程，可分为解析模型、统计模型)来研究，具有很大的灵活性，它可以方便地改变系统结构、参数；而且速度快，可以在很短的时间内完成实际系统很长时间的动态演变过程；精确度高，可以根据需要改变仿真的精度；重复性好，可以很容易地再现仿真过程。

(3) 半实物仿真：又称数学物理仿真或者混合仿真。为了提高仿真的可信度或者针对一些难以建模的实体，在系统研究中往往把数学模型、物理模型和实体结合起来组成一个复杂的仿真系统，这种在仿真环节中存在实体的仿真称为半实物仿真或者半物理仿真。这样的仿真系统有飞机半实物仿真、射频制导导弹半实物仿真等，并且许多模拟器也属于半实物仿真。

按照仿真系统与实际系统时间尺度上的关系，又可将其分为如下几类：

(1) 实时仿真：仿真时钟与系统实际时钟完全一致。许多仿真应用需要满足实时性，这时往往需要实时操作系统或者专用实时仿真硬件的支持。

(2) 欠实时仿真：仿真时钟比实际时钟慢。当对仿真的实时性没有严格的要求时，仿真时钟比实际时钟慢，不影响仿真的目的，采取欠实时仿真则可节约很多资金。

(3) 超实时仿真：仿真时钟比实际时钟快。当实际系统周期太长时，若采用实际时钟就变得毫无意义，这时就要进行超实时仿真。

3. 计算机仿真

计算机仿真是在研究系统过程中根据相似原理，利用计算机来逼真模拟研究对象。研究对象可以是实际的系统，也可以是设想中的系统。在没有计算机以前，仿真都是利用实物或者它的物理模型来进行研究的，即物理仿真。物理仿真的优点是直接、形象、可信，缺点是模型受限、易破坏、难以重用。

计算机作为一种最重要的仿真工具，已经推出了模拟机、模拟数字机、数字通用机、仿真专用机等各种机型并应用在不同的仿真领域。除了计算机这种主要的仿真工具外还有两类专用仿真器：一类是专用物理仿真器，如在飞行仿真中得到广泛应用的转台，各种风洞、水洞等；另一类是用于培训目的的各种训练仿真器，如培训原子能电站、大型自动化工厂操作人员和训练飞行员、宇航员的培训仿真器、仿真工作台和仿真机舱等。

1.1.1 仿真的作用

仿真技术具有很高的科学研究价值和巨大的经济效益。由于仿真技术的特殊功效，特别是安全性和经济性，使得仿真技术得到广泛的应用。首先由于仿真技术在应用上的安全性，使得航空、航天、核电站等成为仿真技术最早的和最主要的应用领域。

归纳起来，仿真技术的主要用途有如下几点：

(1) 优化系统设计。在实际系统建立以前，通过改变仿真模型结构和调整系统参数来优化系统设计。如控制系统、数字信号处理系统的设计经常要靠仿真来优化系统性能。

(2) 系统故障再现，发现故障原因。实际系统故障的再现必然会带来某种危害性，这样做是不安全的和不经济的，利用仿真来再现系统故障则是安全的和经济的。

(3) 验证系统设计的正确性。

(4) 对系统或其子系统进行性能评价和分析。多为物理仿真，如飞机的疲劳试验。

(5) 训练系统操作员。常见于各种模拟器，如飞行模拟器、坦克模拟器等。

(6) 为管理决策和技术决策提供支持。

1.1.1 仿真算法和仿真软件

1. 仿真算法

在建立系统的数学模型后，需要将其转变成能够在计算机上运行的仿真模型。由于计算机只能进行离散的数值计算，因而必须推导出连续系统的递推数学公式，如解微分方程的龙格库塔算法。这实际上属于数值计算的内容，其发展已经相当完善了。其实这就是计算机仿真算法的设计，即把数学模型转化为能在计算机上运行的仿真模型。

通常这些仿真算法并不需要仿真人员去编制，因为这些仿真算法往往已经内嵌于各种面向仿真用途的专用软件中了。但是对这些算法的了解无疑有助于用户更好地完成仿真任务。一般来说，系统仿真算法有如下几类：

(1) 集中参数系统仿真算法。

(2) 分布参数系统仿真算法。

(3) 离散时间系统仿真算法。

2. 仿真软件

仿真软件是一类面向仿真用途的专用软件，它可能是面向通用的仿真，也可能是面向某个领域的仿真。它的功能可以概括为以下几点：

(1) 为仿真提供算法支持。

(2) 模型描述，用来建立计算机仿真模型。

(3) 仿真实验的执行和控制。

(4) 仿真数据的显示、记录和分析。

(5) 对模型、实验数据、文档资料和其它仿真信息的存储、检索和管理(即用于仿真数据信息管理的数据库系统)。

根据软件功能，仿真软件可分为以下三个层次：

(1) 仿真程序库：由一组完成特定功能的程序组成的集合，专门面向某一问题或某一领域。它可能是用通用的语言(C++、FORTRAN等)开发的程序软件包，也可能是依附于某种集成仿真环境的函数库或模块库。

(2) 仿真语言：仿真语言多属于面向专门问题的高级语言，它是针对仿真问题，在高级语言的基础上研制的。

(3) 集成仿真环境：它是一组用于仿真的软件工具的集合，包括设计、分析、编制系统模型，编写仿真程序，创建仿真模型，运行、控制、观察仿真实验，记录仿真数据，分析仿真结果，校验仿真模型等。

1.1.1计算机仿真的一般过程

计算机仿真的一般过程可以表述如下：

(1) 描述仿真问题，明确仿真目的。

(2) 项目计划、方案设计与系统定义。根据仿真目的确定相应的仿真结构(实时仿真还是非实时仿真，纯数学仿真还是半物理仿真等)，规定相应仿真系统的边界条件与约束条件。

(3) 数学建模：根据系统的先验知识、实验数据及其机理研究，按照物理原理或者采取系统辨识的方法，确定模型的类型、结构及参数。注意要确保模型的有效性和经济性。

(4) 仿真建模：根据数学模型的形式、计算机类型、采用的高级语言或其它仿真工具，将数学模型转换成能在计算机上运行的程序或其他模型，也即获得系统的仿真模型。

(5) 试验：设定实验环境/条件和记录数据，进行实验，并记录数据。

(6) 仿真结果分析：根据实验要求和仿真目的对实验结果进行分析处理(整理及文档化)。

仿真问题和仿真目的描述

根据仿真问题确定仿真方案

修改方案

系统描述

建立系统数学模型

修改模型

建立仿真数学模型

编写仿真程序

修改程序

验证模型

仿真目的是否达到要求？模型是否合理？

否

程序有问题？

是

否

建模有问题？

是

否

方案有问题？

是

是

进行仿真研究

图1.1 计算机仿真流程图



1.2 动态系统的Simulink仿真

1.1.1Simulink 简介

Simulink是一个用来对动态系统进行建模、仿真和分析的软件包。使用Simulink来建模、分析和仿真各种动态系统(包括连续系统、离散系统和混合系统)，将是一件非常轻松的事情。它提供了一种图形化的交互环境，只需用鼠标拖动的方法便能迅速地建立起系统框图模型，甚至不需要编写一行代码。

利用Simulink进行系统的建模仿真，其最大的优点是易学、易用，并能依托MATLAB提供的丰富的仿真资源。这里对Simulink的强大功能进行简单的介绍。

1. 交互式、图形化的建模环境

Simulink提供了丰富的模块库以帮助用户快速地建立动态系统模型。建模时只需使用鼠标拖放不同模块库中的系统模块并将它们连接起来。

2. 交互式的仿真环境

Simulink框图提供了交互性很强的仿真环境，既可以通过下拉菜单执行仿真，也可以通过命令行进行仿真。菜单方式对于交互工作非常方便，而命令行方式对于运行一大类仿真如蒙特卡罗仿真非常有用。

3. 专用模块库(Blocksets)

作为Simulink建模系统的补充，MathWorks公司还开发了专用功能块程序包，如DSP Blockset和Communication Blockset等。通过使用这些程序包，用户可以迅速地对系统进行建模、仿真与分析。更重要的是用户还可以对系统模型进行代码生成，并将生成的代码下载到不同的目标机上。

表1.2 Simulink的部分软件工具包

DSP Blockset	数字信号处理工具包
Fixed-Point Blockset	定点运算控制系统仿真工具包
Power System Blockset	电力电动系统工具包
Dials & Gauges Blockset	交互图形和控制面板设计工具包
Communications Blockset	通讯系统工具包
CDMA Reference Blockset CDMA	CDMA通讯系统设计和分析工具包
Nonlinear Control Design Blockset	非线性控制设计工具箱
Motorola DSP Developer’s Kit	Motorola DSP开发工具箱
TI DSP Developer’s Kit	TI DSP开发工具箱

4. 提供了仿真库的扩充和定制机制

Simulink的开放式结构允许用户扩展仿真环境的功能：采用MATLAB、FORTRAN和C代码生成自定义模块库，并拥有自己的图标和界面。因此用户可以将使用FORTRAN或C编写的代码链接进来，或者购买使用第三方开发提供的模块库进行更高级的系统设计、仿真与分析。

5. 与MATLAB工具箱的集成

由于Simulink可以直接利用MATLAB的诸多资源与功能，因而用户可以直接在Simulink下完成诸如数据分析、过程自动化、优化参数等工作。工具箱提供的高级的设计和分析能力可以融入仿真过程。

简而言之，Simulink具有以下特点：

(1) 基于矩阵的数值计算。

(2) 高级编程语言。

(3) 图形与可视化。

(4) 工具箱提供面向具体应用领域的功能。

(5) 丰富的数据 I/O 工具。

(6) 提供与其它高级语言的接口。

(7) 支持多平台(PC / Macintosh / UNIX)。

(8) 开放与可扩展的体系结构。

1.1.1 Simulink的应用领域

至此，读者应该对动态系统的模型建立、系统仿真与分析有了一个比较感性的认识；同时对Simulink的强大功能也会有一定的了解。那么使用Simulink到底可以对什么样的动态系统进行仿真分析与辅助设计呢？其实，任何使用数学方式进行描述的动态系统都可以使用Simulink进行建模、仿真与分析。

由于Simulink具有强大的功能与友好的用户界面，因此它已经被广泛地应用到诸多领域之中，如：

(1) 通讯与卫星系统。

(2) 航空航天系统。

(3) 生物系统。

(4) 船舶系统。

(5) 汽车系统。

(6) 金融系统。

此外，Simulink在生态系统、社会和经济等领域也都有所应用。在科学技术飞速发展的21世纪，Simulink的应用领域也将会更加广泛。图1.2所示为Simulink在一些领域中的典型应用。

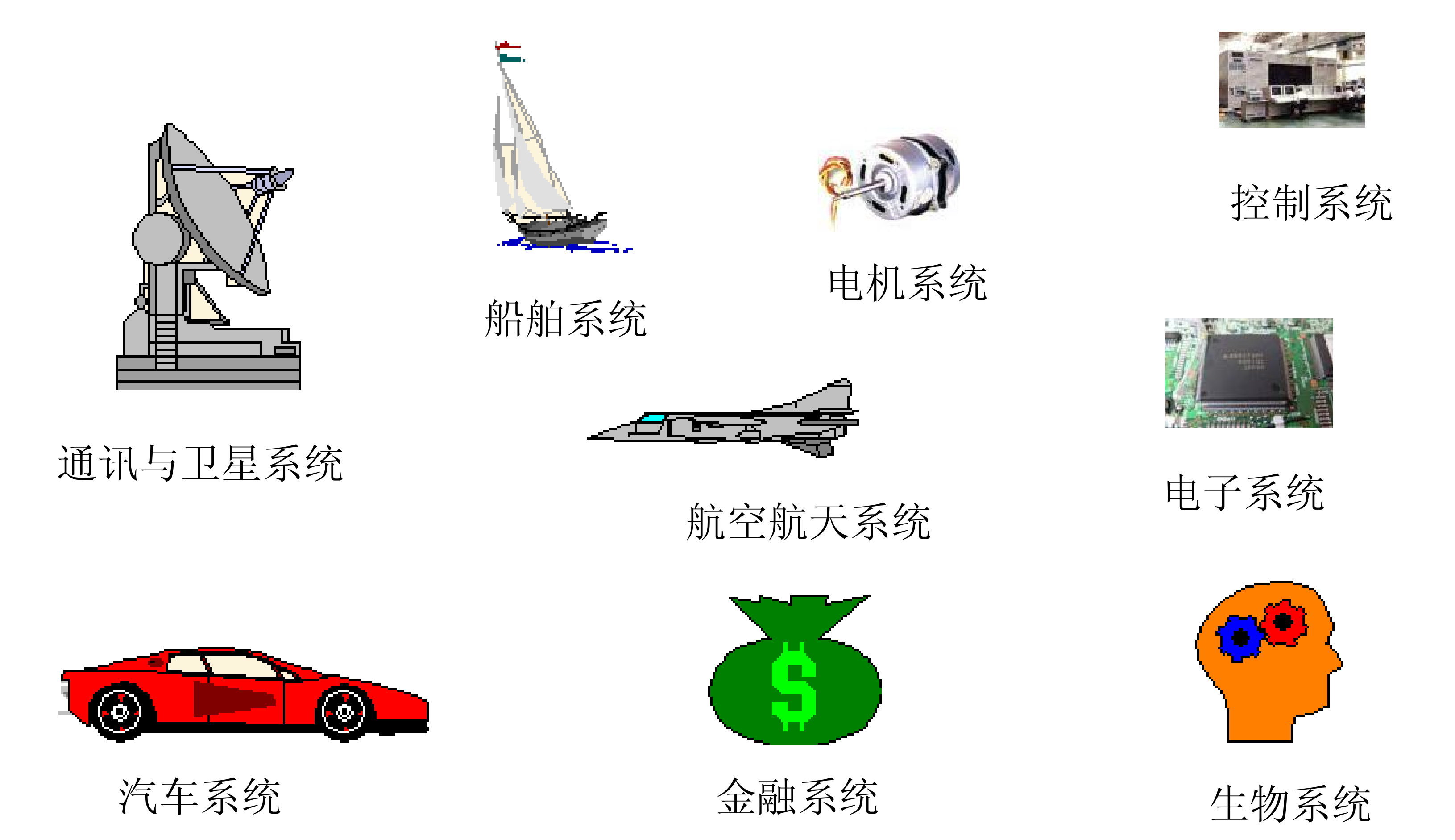


图1.2 Simulink的应用领域示意图

1.1.1 Simulink在MATLAB家族中的位置

MATLAB是一个包含数值计算、高级图形与可视化、高级编程语言的集成化科学计算环境。MATLAB Toolbox提供了面向专业的函数库，扩展了MATLAB的能力。MATLAB Compiler 自动将MATLAB中的M文件转换成C和C++代码，用于独立应用开发。Simulink是一个交互式动态系统建模、仿真和分析工具。Simulink Blockset提供了丰富的专业模块库，广泛地用于控制、DSP、通讯等系统仿真领域。Stateflow是一种利用有限状态机理论建模和仿真事件驱动系统的可视化设计工具，适合用于描述复杂的开关控制逻辑、状态转移图以及流程图等。

Real-Time Workshop 能够从Simulink模型中生成可定制的代码及独立的可执行程序。Stateflow coder能够自动生成状态图的代码，并且能够自动地结合到RTW生成的代码中。图1.3所示为Simulink与MATLAB的层次结构示意图。

图1.3 Simulink与MATLAB之间的层次关系示意图



1.3 本书的组织结构

在对Simulink做进一步的介绍之前，首先对本书的组织结构进行简单的介绍。不同的用户可以根据自己的专业背景、对系统仿真认识的程度以及对使用Simulink进行动态系统仿真掌握的程度来制定本书的使用方法。

在本书的编排之中，我们竭力使每一章的内容在一定程度上都能够自成体系，以方便不同用户的需要。在使用Simulink进行动态系统模型的建立、仿真与分析时，用户可以随时查阅本书的相应内容，以满足特定的需要。本书既可作为学习Simulink的教材，也可作为一本使用手册供读者查阅。

图1.4所示为本书的组织结构说明。

如果你从来没用过MATLAB，那就从这里开始吧，第一部分教你在MATLAB中畅游的基础知识

如果你从前用过MATLAB，但还从未与Simulink谋面，那你就从这里开始吧，你将碰到你的第一次（Simulink仿真经验）

想知道Simulink的引擎和秘密武器吗？那就到这儿来看看吧

想进行系统级的仿真，从这儿进去

第1章：绪论
第2章：Simulink使用基础
第3章：动态系统模型及其Simulink表示
第6章：Simulink系统仿真原理
第7章：Simulink子系统技术
第8章：Simulink命令行仿真技术
第9章：S-函数

入门
进阶
精通
实例分析

图1.4 本书的组织结构说明



第2章　Simulink使用基础



2.1 MATLAB的计算单元：向量与矩阵

MATLAB作为一个高性能的科学计算平台，主要面向高级科学计算。MATLAB的基本计算单元是矩阵与向量，向量为矩阵的特例。一般而言，二维矩阵为由行、列元素构成的矩阵表示；对于m行、n列的矩阵，其大小为$m \times n$。在MATLAB中表示矩阵与向量的方法很直观，下面举例说明。

例如，矩阵 $A=\begin{bmatrix}1 & 2 & 3\\ 4 & 5 & 6\end{bmatrix}$，行向量 $C=\begin{bmatrix}4\\ 5\\ 6\end{bmatrix}$，列向量，在MATLAB中可以分别表示为

```
>>A=[1 2 3; 4 5 6]
>>B=[1 2 3]
>>C=[4; 5; 6]
```

注意：(1) MATLAB中所有的矩阵与向量均包含在中括号[]之中。如果矩阵的大小为1×1，则它表示一个标量，如

>>a=3　　　　　　　　%a表示一个数

(2) 矩阵与向量中的元素可以为复数，在MATLAB中内置虚数单元为i、j；虚数的表达很直观，如3＋4*i或者3＋4*j 。

技巧：(1)　MATLAB中对矩阵或向量元素的引用方式与通常矩阵的引用方式一致，如A(2 ,3)表示矩阵***A***的第2行第3列的元素。如若对***A***的第2行第3列的元素重新赋值，只需键入如下命令:

```
>>A(2,3)=8;
```

则矩阵***A***变为

```
A =
 1    2    3
 4    5    8
```

(2) MATLAB中分号(;)的作用有两点：一是作为矩阵或向量的分行符，二是作为矩阵或向量的输出开关控制符。即如果输入矩阵或向量后键入分号，则矩阵与向量不在MATLAB命令窗口中显示，否则将在命令窗口中显示。如输入矩阵

```
>>A=[1 2 3; 4 5 6]                    % 按下Enter键，则在MATLAB命令窗口中显示
>>A =
        1    2    3
        4    5    6
```

(3) 冒号操作符(:)的应用。冒号操作符在建立矩阵的索引与引用时非常方便且直接。如上述对多维矩阵***F***的建立中，冒号操作符表示对矩阵***F***第一维与第二维所有元素按照其顺序进行引用，从而对***F***进行快速赋值，无需一一赋值。如

```
>>B=2:5                              %对向量进行赋值
>>B=
 2  3  4  5
>>B（1:3）=2                         %向量B从第1个到第3个元素全部赋值为2
```

>>B=

2 2 2 5

>> C=6: -2:0　　　%将向量C进行递减赋值，初始值为6，终止值为0，步长为－2

>>C=

6 4 2 0　　冒号操作符的使用很灵活，如图2.1所示。

a= [1 2 3; 4 5 6; 7 8 9]

a(1,1)

a([1 2 3],3)
a(1:3,3)
a(:,3)

a(2:3,1:2)

图2.1 冒号操作符使用示意图

从图2.1中可以看出，使用冒号操作符对矩阵元素进行引用非常灵活和方便，它可以有效地对矩阵的指定元素或指定区域进行各种操作与控制。这使得MATLAB对矩阵的操作方式非常符合习惯的用法，易于理解与应用。此外，MATLAB还支持多种不同类型的数据，它们的建立和基本引用与本节介绍的基本相同，这里不再赘述。有兴趣的读者可以参考MATLAB的联机帮助。



2.2 MATLAB计算单元的基本操作

2.1节介绍了MATLAB的基本计算单元，即矩阵与向量的建立与引用方法。本节将简单介绍在MATLAB环境下矩阵与向量的操作与运算。

1. 矩阵加法与减法

如果矩阵***A***与矩阵***B***具有相同的维数，则可以定义矩阵的加法与减法，其结果为矩阵相应元素作运算所构成的矩阵。矩阵加法与减法在MATLAB中的表达方式为

```
>> C=A+B;                    %C为矩阵A与B之和
>> D=A-B;                    %D为矩阵A与B之差
```

【例2.1】 若，，则 $A=\begin{bmatrix}1 & 2 & 3\\4 & 5 & 6\end{bmatrix}$ $B=\begin{bmatrix}0 & 2 & 1\\2 & 5 & 3\end{bmatrix}$

则

>>C=

1 4 4

6 10 9

>>D=

1 0 2

2 0 3

矩阵与标量的加法与减法是指标量本身与矩阵所有元素进行相应运算，如若$b=1$，$\boldsymbol{E}=\boldsymbol{A}+\boldsymbol{b}$，则

```
>>E=
   2   3   4
   5   6   7
```

2. 矩阵的乘法与除法

如果矩阵***A***的列数等于矩阵***B***的行数，则矩阵***A***、***B***可以相乘。其结果***C=AB***在MATLAB中可表示为

```
>>C=A*B;                    %A、B相乘，若A、B不满足矩阵乘法法则，MATLAB会给出出错信息
```

【例2.2】 若，，则 $A=\begin{bmatrix}1 & 2 & 3\\4 & 5 & 6\end{bmatrix}$ $B=\begin{bmatrix}1 & 2\\0 & 3\\2 & 1\end{bmatrix}$

则

>>C=

7　11

16　29

如果矩阵***A***为方阵，***A***的***p***次方可以用***A***^***p***表示。如果***p***是一个正整数，那么这个幂可以由矩阵的连续相乘定义。当***p***=0时，其结果为与***A***相同的矩阵；当***p***<0时，只有在***A***的逆存在时才可定义***A***^***p***，其意义为inv(***A***)^(－***p***)。

在MATLAB中，矩阵除法有两种形式，即左除(\)和右除(/)。如果***A***是一个非奇异方阵，那么

>>A\B % 表示A的逆与B的左乘，即inv(A)*B

>>B/A　　% 表示A的逆与B的右乘，即B* inv(A)

矩阵的左除和右除运算还可以用来求解矩阵方程***AX=B***的解：

>>X=A\B

如果***A***是一个方阵，***X***就是方程的解；如果***A***是一个行数大于列数的矩阵，***X***就是方程的最小二乘解。

3. 矩阵的转置

转置是一种重要的矩阵运算，在MATLAB中由撇号表示:

```
>> B=A‘                 % B为A的转置
```

如果 $\boldsymbol{A}=\begin{bmatrix}1 & 2 & 3\\4 & 5 & 6\end{bmatrix}$，则 $\boldsymbol{B}=\begin{bmatrix}1 & 4\\2 & 5\\3 & 6\end{bmatrix}$。如果***A***中含有复数元素，则***A***的转置矩阵中的元素为原来元素的共轭。

4. 对矩阵元素的操作与运算

在上述各种常用运算中，所有的操作都是针对矩阵所有元素或一部分元素的操作。其实还可以对矩阵元素进行单独的操作运算。对于加法和减法，对矩阵元素的操作与对矩阵的操作是一致的。其它运算对于所有矩阵元素的操作需要在操作符前加点，如【例2.3】所示。

【例2.3】 若 $\boldsymbol{A}=\begin{bmatrix}1 & 2\\ -1 & 5\end{bmatrix}$ $\boldsymbol{B}=\begin{bmatrix}7 & 2\\ 1 & 0\end{bmatrix}$ $\boldsymbol{C}=\begin{bmatrix}1+2\mathrm{i} & 5-2\mathrm{i}\\ 3+\mathrm{i} & 1+3\mathrm{i}\end{bmatrix}$

则

```
>>A.*B=                              % 矩阵对应元素相乘
7    4
-1   0
>>B./A=                              % 矩阵对应元素相除
7    1
-1   0
```

```
>>B.^2=                    % 矩阵元素乘方运算
49   4
1    0
>>A.^B=                    % 矩阵对应元素幂运算
1   4
-1  1
>>C.'=           % 矩阵转置
1.0000+2.0000i,3.0000+1.0000i
5.0000-2.0000i,1.0000+3.0000i
```

作为高性能的科学计算工具，MATLAB对矩阵与向量提供强大的支持；但由于本书主要讲述使用Simulink需要具备的MATLAB的基础知识，因此对这部分内容仅作简单介绍，感兴趣的读者可以参考任何一本MATLAB参考书。



2.3 多项式表达与基本运算

Simulink用于动态系统建模、仿真与分析时，将会大量使用多项式。许多系统的模型描述(如系统的传递函数)都需要使用多项式，并在多项式描述的基础上对系统进行仿真分析。本节将简单介绍MATLAB中的多项式表示及其基本运算。

1. 多项式的建立

在MATLAB中，n阶多项式$p(x)$由一个长度为$n+1$的向量$\boldsymbol{p}$所表示，向量$\boldsymbol{p}$的元素为多项式的系数，且按照自变量x的降序排列。若n阶多项式为

$$p(x)=a_n x^n + a_{n-1}x^{n-1} + \cdots + a_2 x^2 + a_1 x + a_0 \quad （其中）a_n \neq 0$$

则其在MATLAB中的表示方法为

$$\boldsymbol{p}=[a_n \quad a_{n-1} \quad \cdots \quad a_2 \quad a_1 \quad a_0]$$

注意，多项式中系数为0的项不能忽略，$\boldsymbol{p}$中相应元素应置为0。如多项式在MATLAB中应表示为

```
>>p=[3   0   2   3]
```

2. MATLAB中多项式操作函数简介

(1) roots(p)：长度为n的向量，表示n阶多项式的根，即方程p(x)=0的根，可以为复数。

(2) conv(p，q)：表示多项式p，q的乘积，一般也指p，q的卷积。

(3) poly(A)：计算矩阵A的特征多项式向量。

(4) poly(p)：由长度为n的向量中的元素为根建立的多项式，结果是长度为n＋1的向量。

(5) polyval(p，x)：若x为一数值，则计算多项式在x处的值；若x为向量，则计算多项式在x中每一元素处的值。

3. 举例分析

【例2.4】 求多项式的根。

解： 在MATLAB的命令窗口中依次输入以下命令，即可求出其根。

```
>> p=[3  0  2  3];
>>rootp=roots(p)                 % rootp为多项式的根
>>rootp =
   0.3911 + 1.0609i
   0.3911 - 1.0609i
  -0.7822
```



2.4　MATLAB的基本绘图功能

MATLAB作为高性能、交互式的科学计算工具，具有非常友好的图形界面，这使得MATLAB的应用非常广泛；同时MATLAB也提供了强大的绘图功能，这使得用户可以通过对MATLAB内置绘图函数的简单调用，便可迅速绘制出具有专业水平的图形。在利用Simulink进行动态系统仿真时，图形输出可以使设计者快速地对系统性能进行定性分析，故可大大缩短系统开发时间。

MATLAB的图形系统是面向对象的。图形的要素，如坐标轴、标签、观察点等都是独立的图形对象。一般情况下，用户不需直接操作图形对象，只需调用绘图函数就可以得到理想的图形。通过本节的学习，用户能够快速掌握图形绘制技术。

1. 基本的二维图形绘制命令

(1) plot(x，y)：输出以向量x为横坐标，以向量y为纵坐标且按照x，y元素的顺序有序绘制的图形。x与y必须具有相同长度。

(2)　plot(y)：输出以向量*y*元素序号m为横坐标，以向量*y*对应元素y_m为纵坐标绘制的图形。

(3)　plot(x1，y1，'str1'，x2，y2，'str2'，…)：用'str1'指定的方式，输出以x1为横坐标，y1为纵坐标的图形。用'str2'指定的方式，输出以x2为横坐标，y2为纵坐标的图形。若省略'str'，则MATLAB自动为每条曲线选择颜色与线型。'str'选项中的部分参数如表2.1所示。

表2.1 plot命令选项

颜　色		线　型			
'g'	绿色	' . '	粗点线	'--'	虚线
'y'	黄色	': '	点线	'-.'	点划线
'r'	红色	'*'	星线	'-'	实线
'b'	蓝色	'o'	圆圈	'+'	加号
'm'	品红色	'x'	叉	's'	方形
'y'	黄色	'd'	菱形	'p'	五角星
'k'	黑色	'^'	上三角	'h'	六角星

2. 简单的三维图形绘制命令

(1) plot3（x，y，z）：用向量x、y和z的相应点(x_i，y_i，z_i)进行有序绘制三维图形。向量x，y，z必须具有相同的长度。

(2) plot3(x1，y1，z1，'str1'，x2，y2，z2'str2'，…)：用'str1'指定的方式，对x1，y1和z1进行绘图；用'str2'指定的方式，对x2，y2和z2进行绘图；如果省略'str'，则MATLAB自动选择颜色与线型。

3. 图形绘制举例

【例2.6】　用MATLAB绘制正弦函数在[0，2π]中的图形。

解：在MATLAB命令行下输入

```
>> x=0:0.1:2*pi;      % pi为MATLAB中默认的圆周率
>> y=sin(x);
>> plot(x, y, '*');
```

其中x为自变量，这里使用冒号表达式设定其取值步长为0.1，取值范围为[0，2π]。用星号'*'输出图形，结果如图2.2所示。

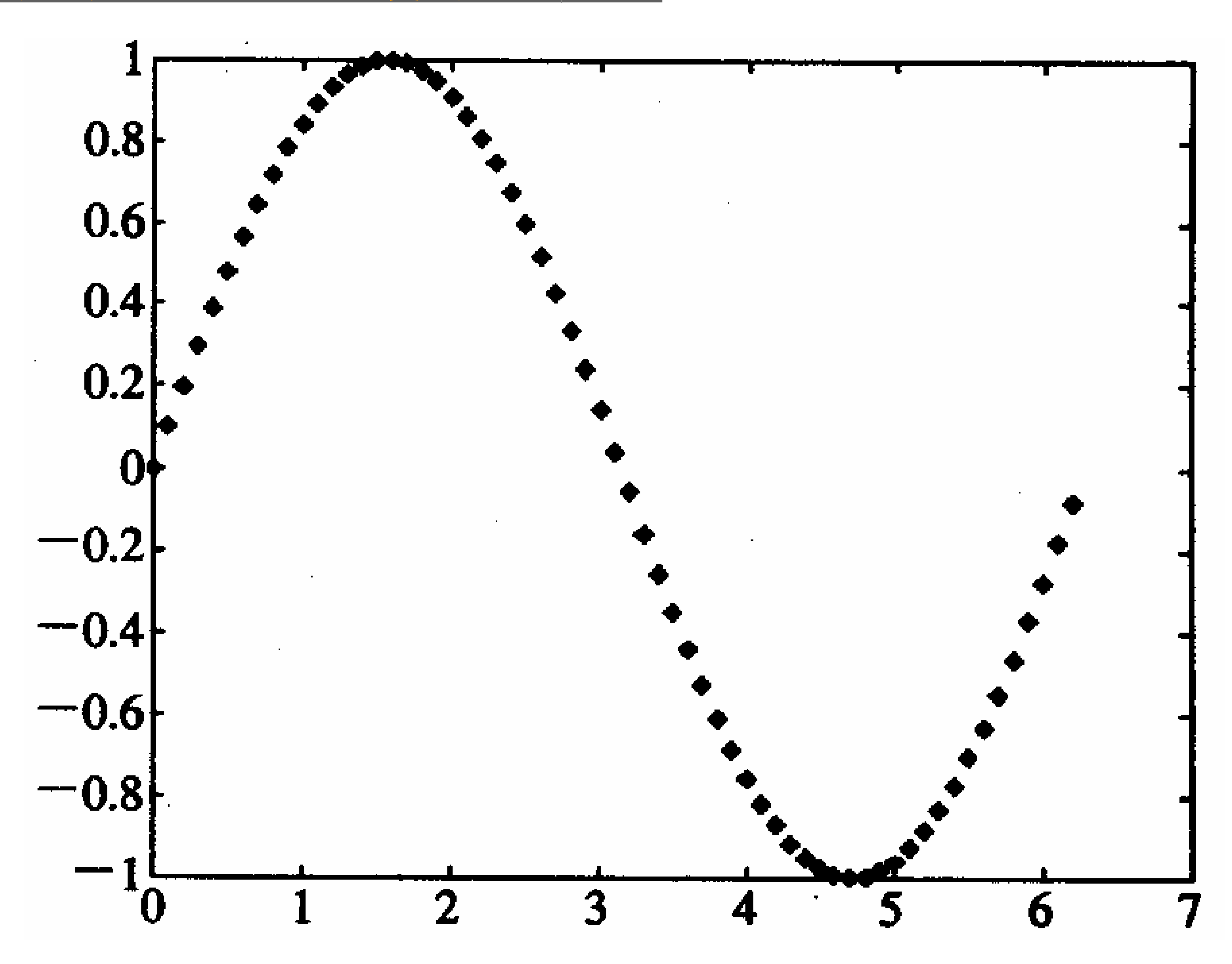


图2.2 绘制正弦函数曲线

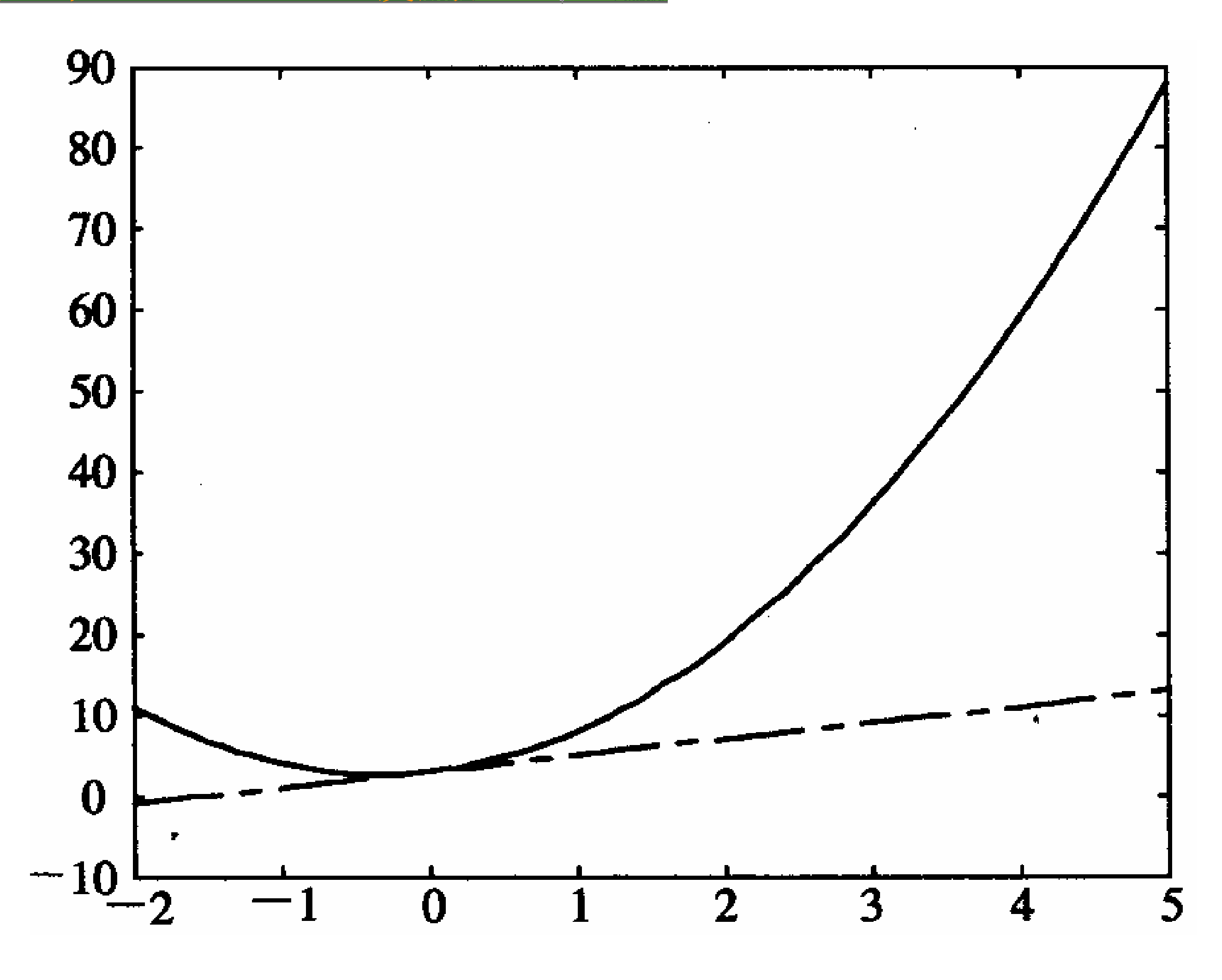


图2.3 多项式绘制

4. 简单的图形控制命令

(1) clc：清除命令窗口。

(2) grid：自动在各个坐标轴上加上虚线型的网格。

(3) hold on：保持当前的图形，允许在当前图形状态下绘制其它图形，即在同一图形窗口中绘制多幅图形。

(4) hold off：释放当前图形窗口，绘制的下一幅图形将作为当前图形，即覆盖原来图形。这是MATLAB的缺省状态。

(5) hold：在hold on与hold off之间进行切换。

5. 简单的子图命令

(1) subplot(m,n,p)：将图形窗口分成m行n列的子窗口，序号为p的子窗口为当前窗口。子窗口的编号由上至下，由左至右。

(2) subplot：设置图形窗口为缺省模式，即subplot(1，1，1)单窗口模式。

【例2.8】 绘出在三维空间中的一个随机曲线。

解： 在MATLAB命令行下输入

```
>> x=cumsum(rand(1,1000)-0.5);
>> y=cumsum(rand(1,1000)-0.4);
>> z=cumsum(rand(1,1000)-0.3);
>> plot3(x,y,z)
>> grid;
```

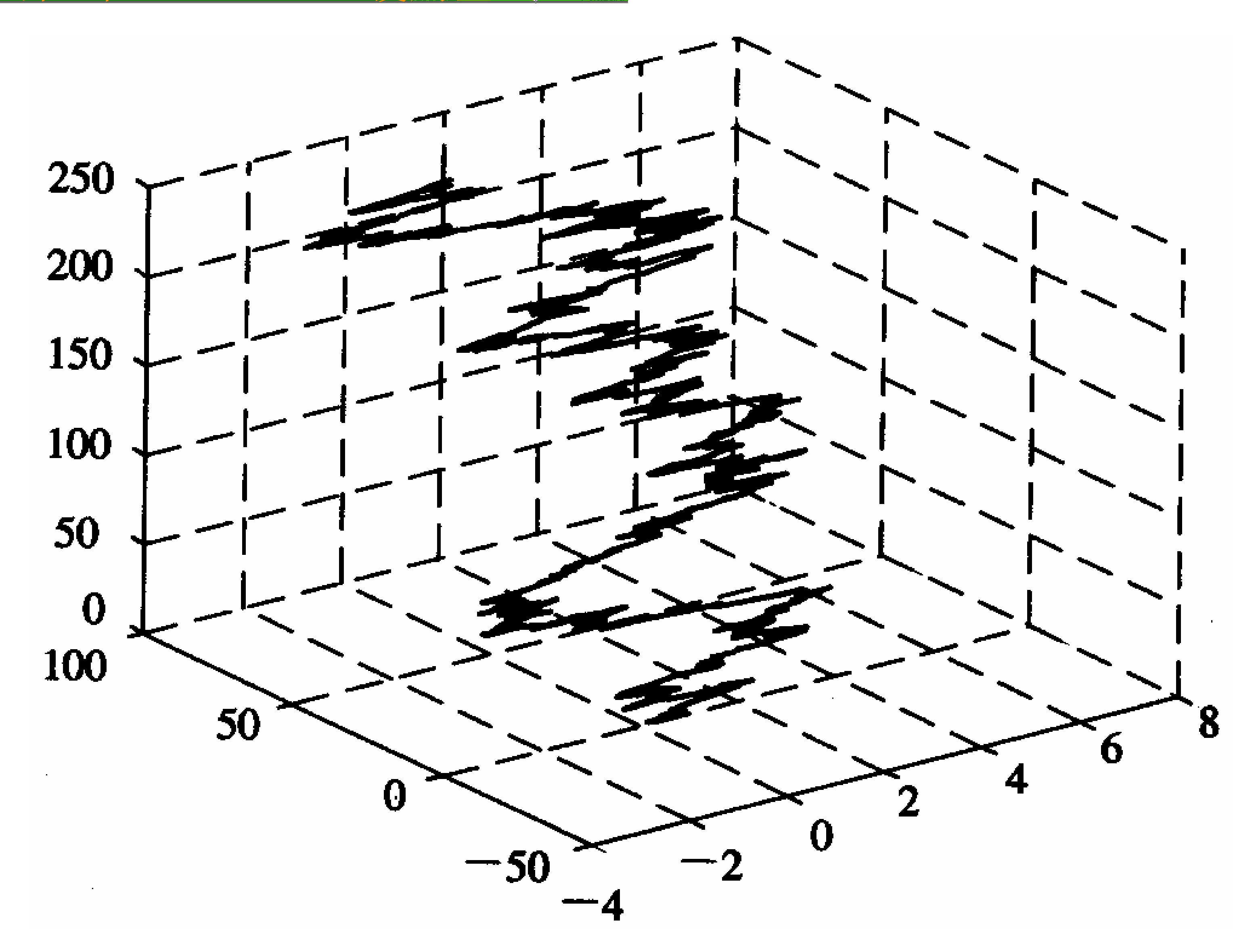


图2.4 随机曲线绘制

【例2.9】 在一个图形窗口的左侧子图中绘制函数，$y_1(x)=x^3-2x-3$ 在右侧子图中绘制函数 $y_2(x)=x\sin(x)$

其中 $x\in[-3,3]$ 。

解：在MATLAB命令行下输入：

```
>> x=-3:0.1:3;
>> y1=x.^3-2*x-3;
>> y2=x.*sin(x);
>> subplot(1,2,1),plot(x,y1,'*'),grid
>> subplot(1,2,2),plot(x,y2,'-'),grid
```

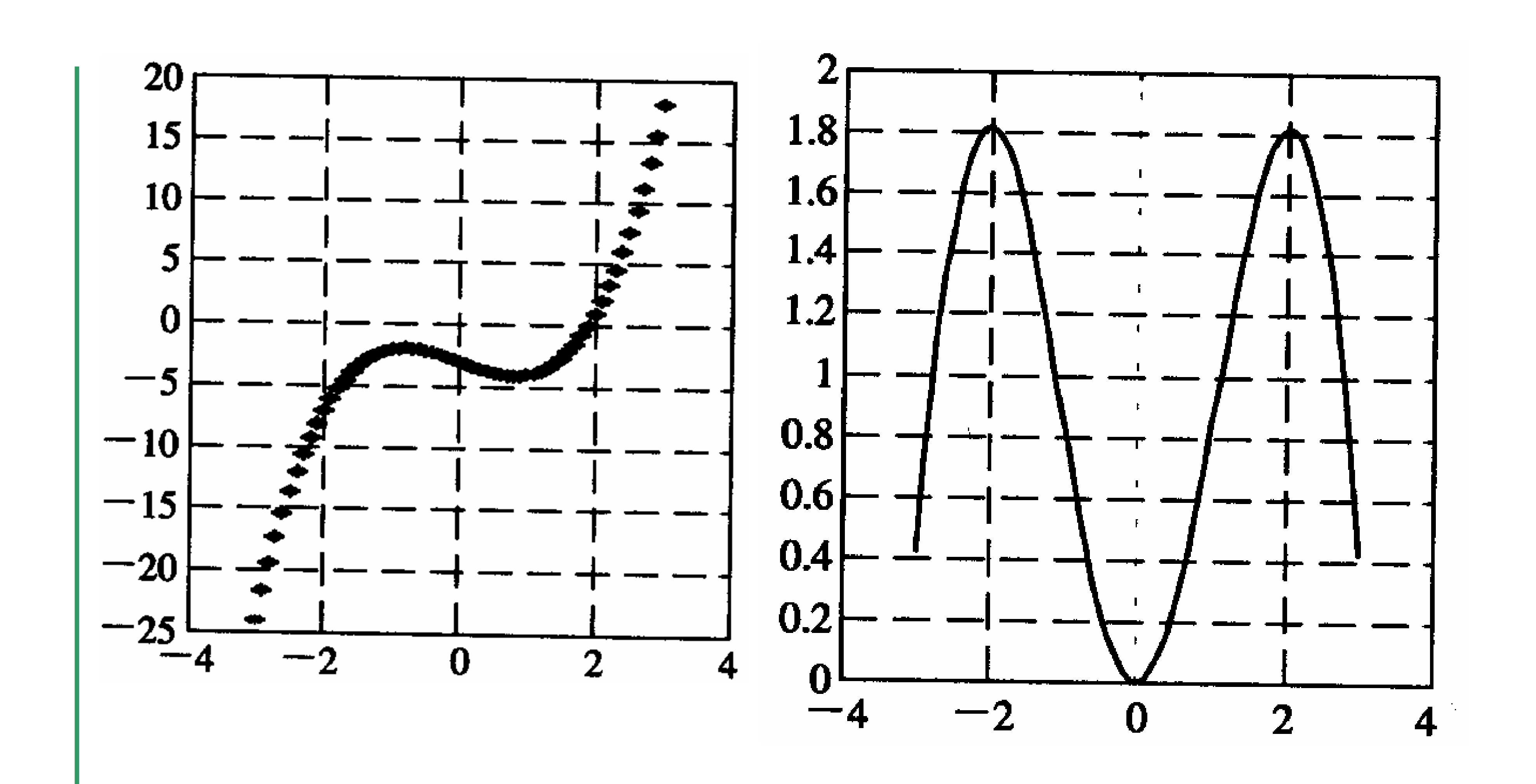


图2.5　子图绘制

由此可见，MATLAB的图形绘制功能非常强大，文中仅以几个简单的例子说明，读者可以进一步对生成的图形进行更低层操作(操作图形对象)，以便获得更好的效果，这里不再赘述。



2.5 M文件与MATLAB函数

2.1.1 M文件编辑器

“工欲善其事，必先利其器。”用户应首先熟悉一下最经常使用的M文件编辑器(M File Editor)。M文件编辑器不仅仅是一个文字编辑器，它还具有一定的程序调试功能，虽然没有像VC、BC那样强大的调试能力，但对于调试一般不过于复杂的MATLAB程序已经足够了。在MATLAB命令行下输入

```
>>edit
```

则弹出如图2.6所示的M文件编辑器窗口。

D:\MATLAB6p1\work\demofun.m

File Edit View Text Debug Breakpoints Web Window Help

```
function y=demofun(x)          %设定自变量x的取值范围
leng=length(x);                %计算向量x的长度
for m=1:leng                   %计算函数值
    if x(m)<=0                 %判断x取值所在范围
        y(m)=sin(x(m));        %计算分段函数值
    elseif x(m)<=3
        y(m)=x(m);             %计算分段函数值
    else
        y(m)=-x(m)+6;          %计算分段函数值
    end
end
```

Ready

图2.6 M文件编辑器

1. 编辑功能

(1) 选择：与通常鼠标选择方法类似，但这样做其实并不方便。如果习惯了，使用Shift+箭头键是一种更为方便的方法，熟练后根本就不需要再看键盘。

(2) 拷贝粘贴：没有比Ctrl+C、Ctrl+V键更方便的了，相信使用过Windows的人一定知道。

(3) 寻找替代：寻找字符串时用Ctrl+F键显然比用鼠标点击菜单方便。

(4) 查看函数：阅读大的程序常需要看看都有哪些函数并跳到感兴趣的函数位置，M文件编辑器没有为用户提供像VC或者BC那样全方位的程序浏览器，却提供了一个简单的函数查找快捷按钮，单击该按钮，会列出该M文件所有的函数。

(5) 注释：如果用户已经有了很长时间的编程经验而仍然使用Shift+5来输入%号，一定体会过其中的痛苦（忘了切换输入法状态时，就会变成中文字符集的百分号）。

(6) 缩进：良好的缩进格式为用户提供了清晰的程序结构。编程时应该使用不同的缩进量，以使程序显得错落有致。增加缩进量用Ctrl+]键，减少缩进量用Ctrl+[键。当一大段程序比较乱的时候，使用smart indent (聪明的缩进，快捷键Ctrl+I)也是一种很好的选择。

2. 调试功能

M程序调试器的热键设置和VC的设置有些类似，如果用户有其它语言的编程调试经验，则调试M程序显得相当简单。因为它没有指针的概念，这样就避免了一大类难以查找的错误。不过M程序可能会经常出现索引错误，如果设置了stop if error(Breakpoints菜单下)，则程序的执行会停在出错的位置，并在MATLAB命令行窗口显示出错信息。下面列出了一些常用的调试方法。

(1) 设置或清除断点：使用快捷键F12。

(2) 执行：使用快捷键F5。

(3) 单步执行：使用快捷键F10。

(4) step in：当遇见函数时，进入函数内部，使用快捷键F11。

(5) step out：执行流程跳出函数，使用快捷键Shift+F11。

(6) 执行到光标所在位置：非常遗憾这项功能没有快捷键，只能使用菜单来完成这样的功能。

(7) 观察变量或表达式的值：将鼠标放在要观察的变量上停留片刻，就会显示出变量的值，当矩阵太大时，只显示矩阵的维数。

(8) 退出调试模式：没有设置快捷键，使用菜单或者快捷按钮来完成。

2.1.1 MATLAB语言的语法

1. 注释

MATLAB中用百分号%表示其后为程序注释(实际上在前面已经碰到了这样的注释)。编写M程序和编写其它程序一样应该养成良好的程序注释习惯。除了程序间的注释，编写M文件时还应该在文件头说明该程序的功能和使用方法，使用Help命令看到的帮助信息正是这些在文件头的注释。

2. 赋值语句

在MATLAB中，赋值语句的基本语法结构为

variablename=value；

3. 逻辑表达式

在MATLAB中，逻辑表达式的基本语法结构为

logicalvalue＝variable1 关系运算符 varialble2；

logicalvalue＝logical expression 1 逻辑运算符 logical expression 2；

其中关系运算符有= =(等于)、～＝(不等于)、>(大于)、<(小于)、>=(不小于)、<=(不大于)等。逻辑运算符有&(逻辑与)、|(逻辑或)和~(逻辑非)等。

4. 条件控制语句

MATLAB中由if语句进行判断，其基本语法结构为

```
if 逻辑表达式
语句集合
end
```

在if与逻辑表达式之间必须有一个空格；当逻辑表达式值为真时，执行语句集合中的语句；这里语句集合可以是MATLAB中的单独命令，也可以是由逗号、分号隔开的语句集合或return语句。

对于简单的语句也可以写成下面的形式：

if 逻辑表达式，语句集合，end

此外，if语句还可以与elseif、else组合成更为复杂的控制语句，其语法格式如下：

```
if  逻辑表达式
语句集合1
else
语句集合2
end
```

5. 循环语句

MATLAB中实现循环的语句有两种：for语句与while语句，以实现某些语句的循环执行。for语句语法格式如下：

```
for 变量=表达式
语句集合
end
```

2.1.1 MATLAB脚本文件与M函数

MATLAB中有两种M文件：一种称为脚本文件（类似于批处理语句），另一种是M函数（类似于函数的概念）。

1. 脚本文件

脚本文件是由一系列MATLAB的命令、内置函数以及M文件等构成的文件，它可以由一般的编辑器进行编制，其结果保存在相应的M文件中。M脚本文件的实质为命令的集合，在MATLAB中执行M脚本文件时，MATLAB从文件中读取命令执行，完成用户的工作。

一般习惯于使用MATLAB的编辑器编制M文件。打开MATLAB编辑器，新建M脚本文件，保存时系统会自动将文件保存成*.m文件。然后便可以在MATLAB命令窗口或其它M文件中使用。其特点是按照脚本中语句的顺序执行，生成的变量放在当前的工作区之中（如果从命令行运行，则放在基本工作区）。

【例2.10】 编写一个M文件绘制函数 $y(x)=\begin{cases}\sin x, & x\le 0\\ x, & 0<x\le 3\\ -x+6, & x>3\end{cases}$

在区间中的图形。

解：在MATLAB命令行下输入edit命令以打开M文件编辑器，输入以下程序：

```
x=-6:0.1:6;                 % 设定自变量x的取值范围
leng=length(x);             % 计算向量x的长度
```

```
for m=1:leng                         % 计算函数值
    if x(m)<=0                       % 判断x取值所在范围
        y(m)=sin(x(m));              % 计算分段函数值
    elseif x(m)<=3
        y(m)=x(m);                   % 计算分段函数值
    else
        y(m)=-x(m)+6;                % 计算分段函数值
    end
end
plot(x,y,'*'), grid;                 %绘制函数曲线
```

将其存盘为demomfile1.m（该文件就是一个MATLAB脚本文件），然后在MATLAB命令行下输入：

```
>>demomfile1
```

则生成如图2.7所示的函数曲线。

本例目的在于说明M脚本文件的编写技术，以及如何使用前面所讲述的MATLAB语言的条件判断与循环语句。由此可见使用MATLAB语言进行程序设计简单而又快速。

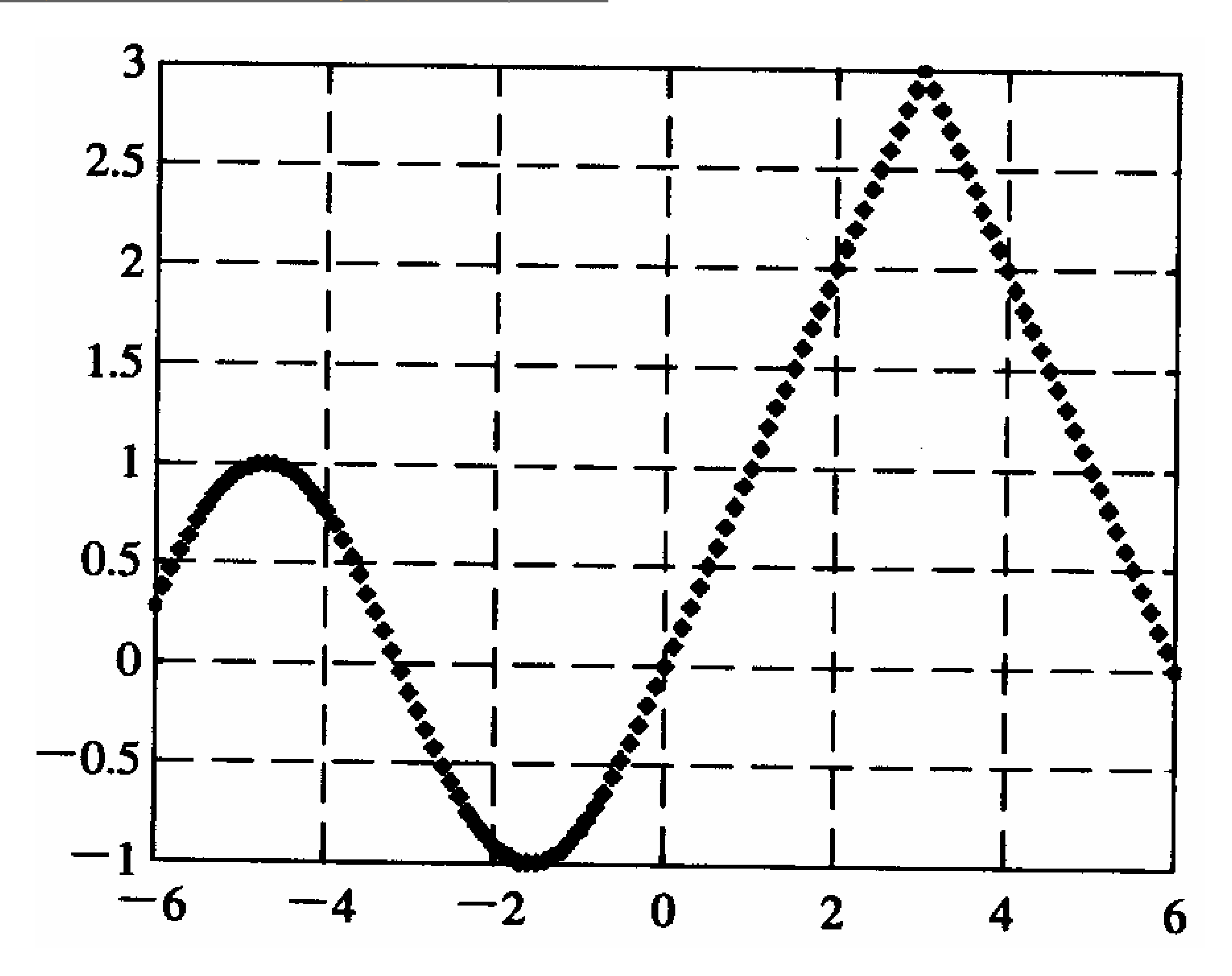


图2.7　使用M文件绘制函数图形

2. M函数

MATLAB的函数与脚本不同，M函数的第一行为关键字function，函数第一次执行时将生成内存代码，生成的变量放在函数的工作区。在MATLAB中有大量的内置函数及大量的工具箱函数，使用它们可以完成大部分的工作；但由于不同的用户有不同的需要，MATLAB允许用户开发自己的专用或通用函数，以扩展MATLAB的函数应用。这里仅简单介绍一下M函数的编制与使用方法。这对理解后面的S-函数有很重要的作用。

(1)　M函数的第一行必须包含function，普通的M文件没有这种要求。

(2) 在function后面必须声明函数名、输入变量（输入参数）与输出变量（输出参数），如function outvar=function_name(inputvar)。

(3)　M函数可以有零个、一个或多个输入或输出。

(4)　M函数的调用方式为：outvar=function_name(inputvar)。

(5) M函数文件名须和函数名function_name相同，调用时函数的输入与输出变量名称不需要和函数定义中的变量相同。

(6) M函数的注释用%开始的行表示，help function_name 显示的是紧接第一行之后的注释。

MATLAB允许将多个函数写在同一个M文件中，其中第一个函数是M文件的主函数，M文件名必须为主函数的名字。其余的函数均为子函数，并受到其它函数的调用。因此，用户可以书写具有模块化特色的MATLAB函数，但是要注意以下几点：

(1) 所有的子函数只能在同一M文件下调用。

(2) 每个子函数都有自己单独的工作区，必须由调用函数传递合适的参数。

(3) 当子函数调用结束后，子函数的工作区将被清空。

【例2.11】　编写一个通用的M函数求取【例2.10】中函数在任意点的值，并绘制函数在区间中的图形。

解：(1) 编写函数demofun并将其存储在同名M文件demofun.m中。

```
function y=demofun(x)                    % M函数定义
leng=length(x);                          % 计算向量x的长度
for m=1:leng                         % 计算函数值
    if x(m)<=0                       % 判断x取值所在范围
```

```
y(m)=sin(x(m));                    % 计算分段函数值
  elseif x(m)<=3
    y(m)=x(m);                     % 计算分段函数值
  else
    y(m)=-x(m)+6;    % 计算分段函数值
  end
end
```

(2) 编写M脚本文件demofile2.m，绘制函数曲线或在命令行下输入下列命令：

```
x=-6:0.1:6;       % 设定x的取值范围
y=demofun(x);            % 调用函数demofun.m求值
plot(x,y)，grid；         % 输出图形
```

结果如图2.8所示，与前面输出图形一致。

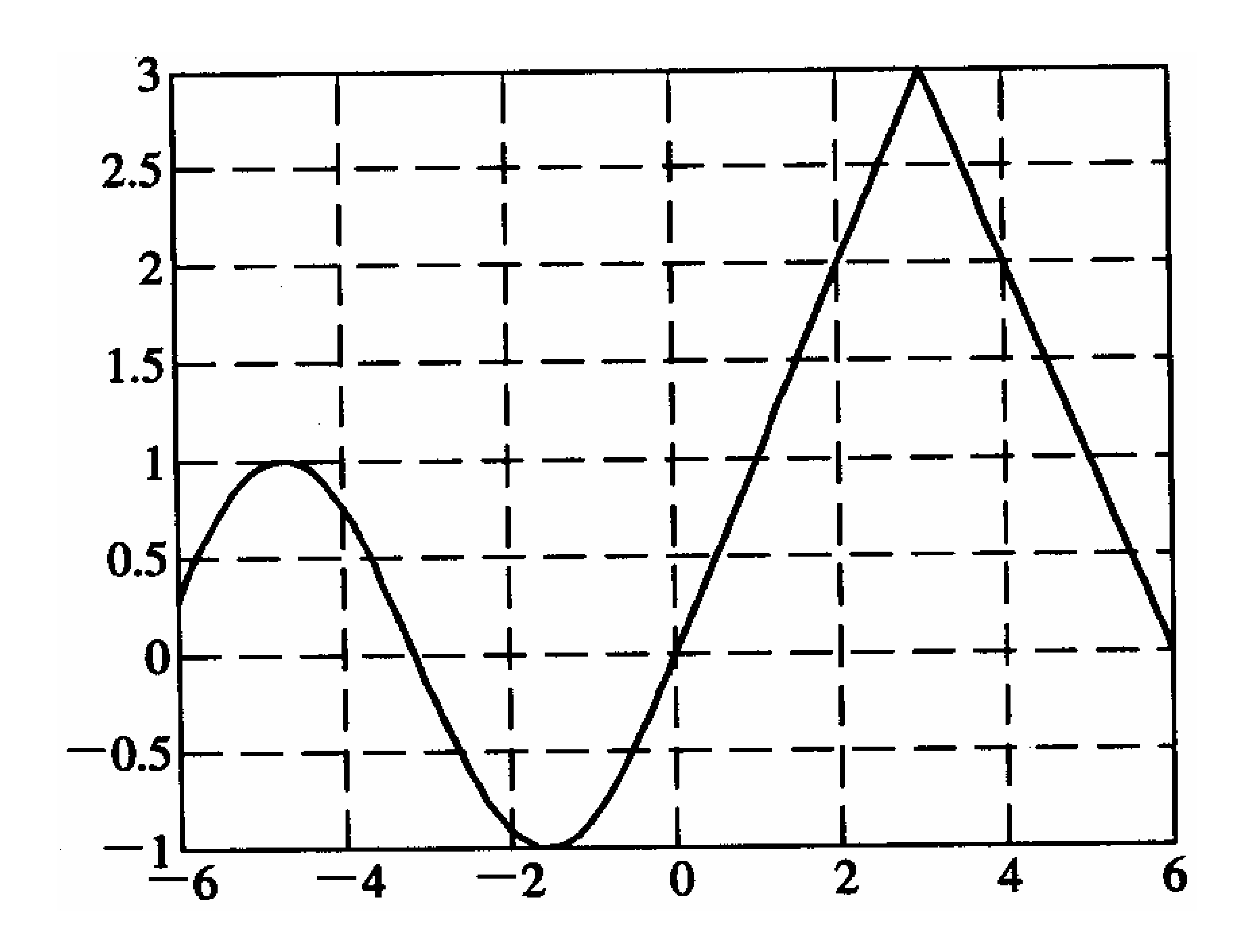


图2.8 使用子函数绘制图形

M函数可以较好地将具有一定功能的脚本文件进行封装，这样有利于程序的阅读与修改。这时可以在MATLAB命令窗口中显示工作区中的变量，输入以下命令：

```
>>whos
  Name Size      Bytes          Class
  x    1×121     968     double array
  y    1×121     968     double array
Grand total is 242 elements using 1936 bytes
```



2.6 MATLAB的单元与结构体

1. 字符串数据

MATLAB作为高性能的科学计算平台，不仅提供高精度的数值计算功能，而且还提供对多种数据类型的支持。如double类型表示双精度浮点数，char表示字符，unit8表示无符号8位整型数等等。除此之外，MATLAB还提供对字符串的支持，在MATLAB中字符串由单引号来定义，如

Strname='Simulation'　% 表示Strname为一字符串，其值为 Simulation

进而可以定义字符（串）矩阵。它与定义普通矩阵类似。

2. 单元矩阵

在前面所提到的矩阵与向量中，矩阵之中所有元素的数据类型均为单一的类型。MATLAB支持复合数据类型的矩阵与向量，这是由一个特殊的矩阵实现的，它就是单元矩阵(Cell类型的矩阵)。在有些书中，单元矩阵也称为细胞矩阵或细胞数组。

单元矩阵的生成方式有如下三种：

(1) 使用花括号{ }直接生成，这与普通矩阵使用中括号[ ]生成方法一致。例如

>>cellmatrix={'xidian'，'press'，20；'xian'，15.21，'university'}；

(2) 直接对单元矩阵中的每一单元分别进行赋值，如

>>cellname{1}='MATLAB'；

>>cellname{2}=20.23;

(3) 通过MATLAB中单元矩阵的创建命令cell创建合适的矩阵。cell的使用方法如下：

>>cellname=cell（m，n） % 表示创建一个m×n的单元矩阵

3. 结构体

如今的程序设计语言中，大都提供了对结构体变量的支持；MATLAB同样支持结构体变量，而且其生成与使用都非常容易、直观。结构体是一个很有用的某些具有某种相关性记录的集合体，它使一系列相关记录集合到一个统一的结构之中，从而使这些记录能够被有效地管理、组织与引用。

在MATLAB中，结构体是按照域的方式生成与存储结构体中的每个记录；一个域中可以包括任何MATLAB支持的数据类型，如双精度数值、字符、单元矩阵及结构等类型。下面简单介绍结构体的生成与引用。

1) 结构体生成

结构体生成方式：

struct_name(record_number).field_name=data;

如某个班级学生花名册的建立：

```
>>student(1).name='Li Yang';
>>student(1).number='0134';
```

```
>>student(2).name='Ma Lei';
>>student(2).number='0135';
...
>>student(33).name='Yao Hui';
>>student(33).number='0166'
```

student是具有33个结构变量的向量，表示某个班级所有33个同学的姓名与学号。每一个记录对应一个学生的姓名与学号。由此可见，在MATLAB中建立结构体是不费吹灰之力的。

2) 结构体引用

在MATLAB中对结构体变量的引用也很简单，如对上述学生花名册中的第二个学生记录的引用如下：

```
>>Name=student(2).name;
>>Number=student(2).number;
```

其结果为

```
Name=
Ma Lei
Number=
0134
```



第3章 动态系统模型及其Simulink表示



3.1 简单系统模型及表示

3.1.1 简单系统的基本概念

不同系统具有不同数量的输入与输出；一般来说，输入输出数目越多，系统越复杂。最简单的系统一般只有一个输入与一个输出，而且任意时刻的输出只与当前时刻的输入有关。本节首先介绍简单系统的基本概念以及简单系统的Simulink表示。

【定义3.1】 **简单系统**。对于满足下列条件的系统，我们称之为简单系统：

(1) 系统某一时刻的输出直接且唯一依赖于该时刻的输入量。

(2) 系统对同样的输入，其输出响应不随时间的变化而变化。

(3) 系统中不存在输入的状态量，所谓的状态量是指系统输入的微分项（即输入的导数项）。

设简单系统的输入为x，系统输出为y，x可以具有不同的物理含义。对于任何系统，都可以将它视为对输入变量x的某种变换，因此可以用T [ ]表示任意一个系统，即

$$y=T[x]$$

对于简单系统，x一般为时间变量或其它的物理变量，并具有一定的输入范围。系统输出变量y仅与x的当前值相关，从数学的角度来看，y是x的一个函数，给出一个x值，便有一个y值与之对应。

【例3.1】 对于如下的一个系统：

$$y=\begin{cases}u^2, & t\in[0,\ 1]\\ u^{\frac{1}{2}}, & t>1\end{cases}$$

其中为系统的输入变量，为时间变量，y为系统的输出变量。输入变量。很显然，此系统服从简单系统的条件，为一简单系统。系统输出仅由系统当前时刻的输入决定。

3.1.1 简单系统的描述方式

一般来讲，简单系统都可以采用代数方程与逻辑结构相结合的方式进行描述。

1. 代数方程

采用数学方程对简单系统进行描述，可以很容易由系统输入求出系统输出，并且由此可方便地对系统进行定量分析。

2. 逻辑结构

一般来说，系统输入都有一定的范围。对于不同范围的输入，系统输出与输入之间遵从不同的关系。由系统的逻辑结构可以很容易了解系统的基本概况。

3.1.1 简单系统的Simulink描述

本章主要介绍动态系统的基本知识，为使用Simulink进行系统仿真打下基础。因此这里并不准备建立系统的Simulink模型，而是采用编写M脚本文件的方式对系统进行描述并进行简单的仿真。下面以【例3.1】中的简单系统为例，说明在Simulink中如何对简单系统进行描述。

【例3.1】中的简单系统，编写如下的systemdemo1.m脚本文件进行描述与分析。

```
% systemdemo1.m文件
u=0:0.1:10;                 % 设定系统输入范围与仿真步长
leng=length(u);             % 计算系统输入序列长度
for i=1:leng                % 计算系统输出序列
    if u(i)<=1              % 逻辑判断
        y(i)=u(i).^2;
    else
         y(i)=sqrt(u(i));
    end
end
plot(u,y);grid;   % 绘制系统仿真结果
```

图3.1 简单系统的输入输出关系图



3.2 离散系统模型及表示

3.1.1 离散系统的基本概念

前面所涉及到的系统中，无论是系统的输入还是系统的输出均是连续的变量，在这里连续指的是系统的输入与输出均在时间变量上连续取值（与数学上函数连续概念并不相同）。本节将简单介绍离散系统的基本概念，系统的描述与简单仿真。

所谓离散系统，是指系统的输入与输出仅在离散的时间上取值，而且离散的时间具有相同的时间间隔。下面给出离散系统更全面的定义。

【定义3.2】**离散系统**。凡是满足如下条件的系统均为离散系统：

(1) 系统每隔固定的时间间隔才“更新”一次，即系统的输入与输出每隔固定的时间间隔便改变一次。固定的时间间隔称为系统的“采样”时间。

(2) 系统的输出依赖于系统当前的输入、以往的输入与输出，即系统的输出是它们的某种函数。

(3) 离散系统具有离散的状态。其中状态指的是系统前一时刻的输出量。

3.1.1 离散系统的数学描述

前面给出了离散系统的定义，这里给出离散系统的数学描述。设系统输入变量为，其中为系统的采样时间，为采样时刻。显然，系统的输入变量每隔固定的时间间隔改变一次。由于为一固定的值，因而系统输入常被简记为。设系统输出为，同样也可简记为。由离散系统的定义可知，其数学描述应为

$$y(n) = f\left(u(n) \quad u(n-1)\,\mathbf{L}\;\mathbf{L}\,; \quad y(n-1) \quad y(n-2)\,\mathbf{L}\;\mathbf{L}\,;\right)$$

【例3.2】 对于如下的离散系统模型：

$$y(n) = u(n)^2 + 2u(n-1) + 3y(n-1)$$

其中系统的初始状态为$y(0)$=3，系统输入为，则系统在时刻0，1，2……的输出分别为

……

$$y(0) = 3$$

$$y(1) = u^2(1) + 2u(0) + 3y(0) = 4 + 0 + 9 = 13$$

$$y(2) = u^2(2) + 2u(1) + 3y(1) = 16 + 4 + 39 = 59$$

离散系统除了采用一般的数学描述方式之外，还可以采用差分方程进行描述。使用差分方程描述方程形式如下：

设系统的状态变量为，离散系统差分方程由以下两个方程构成：

状态更新方程： $x(n+1) = f_d(x(n), u(n), n)$

系统输出方程： $y(n) = g(x(n), u(n), n)$

3.1.1 离散系统的Simulink描述

这里以【例3.2】中的离散系统为例，说明如何利用Simulink对离散系统进行描述，并在此基础上对系统进行简单的分析。与前面相类似，此处并不建立系统的Simulink模型进行仿真，而是编写M脚本文件从原理上对离散系统进行说明，并说明离散系统与连续系统的区别之处。

编写脚本文件systemdemo2.m对【例3.2】中的离散系统进行描述分析。

%systemdemo2.m文件

```
y(1)=3;
% 表示离散系统初始状态为3
% 由于MATLAB中数组下标从1开始，这里y(1)相当于上文中的
   y(0)=3，下同
u(1)=0;                    % 表示离散系统初始输入为0
for i=2:11                 % 设定离散系统输入范围为时刻0到时刻10
  u(i)=2*i;                % 离散系统输入向量
  y(i)=u(i).^2+2*u(i-1)+3*y(i-1);   % 离散系统输出向量
end
plot(u,y);grid;            % 绘制系统仿真结果
```

系统从时刻0到时刻10的输入与输出的关系如图3.2所示。其中横坐标表示离散系统的输入向量，而纵坐标表示离散系统的输出向量。说明：这里并没有指定离散系统的采样时间，而仅仅举例说明离散系统的求解分析。在实际的系统中，必须指定系统的采样时间，只有这样才能获得离散系统真正的动态性能。

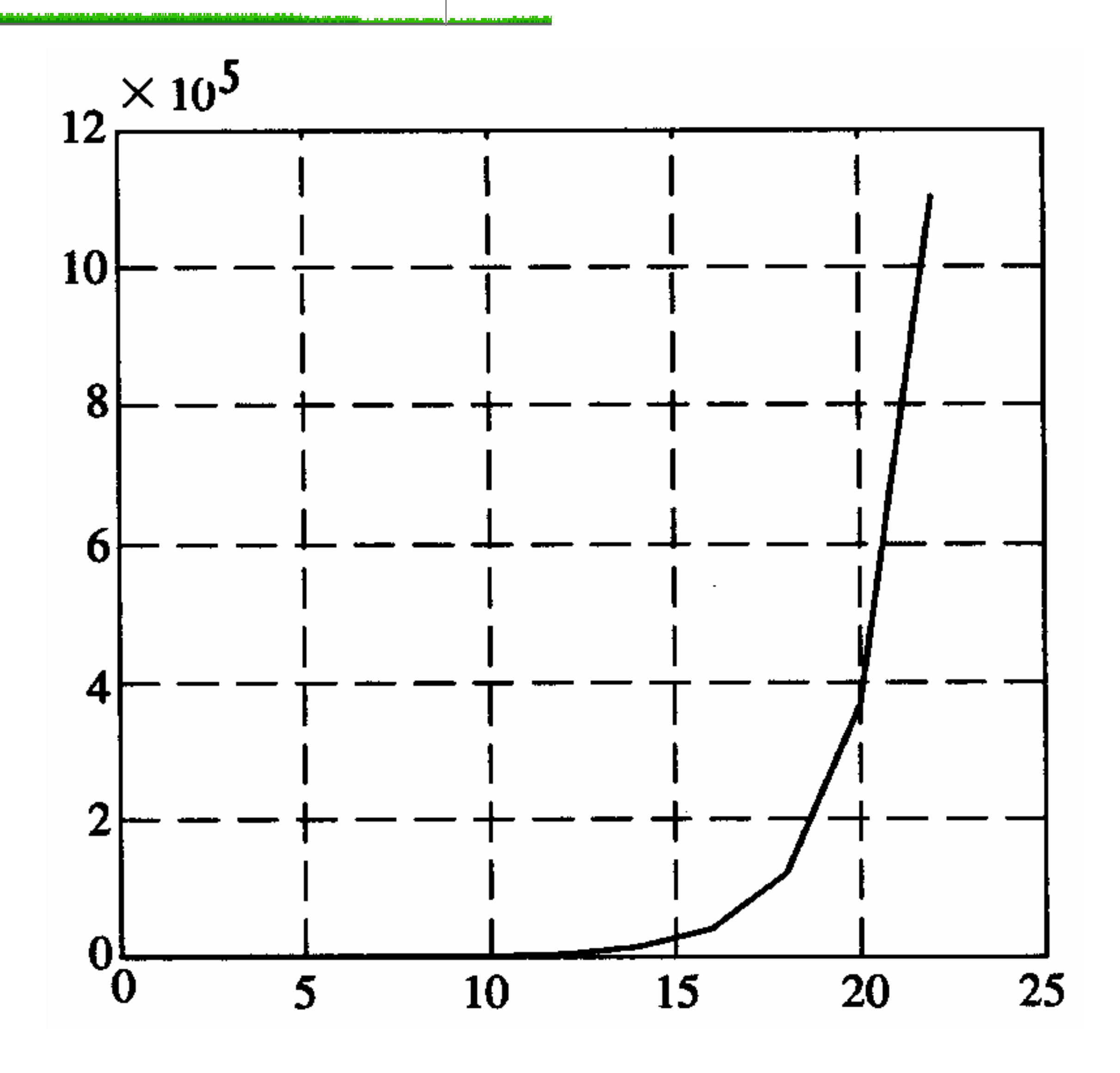


图3.2 【例3.2】中离散系统的输入输出关系

3.1.1线性离散系统

对于任何系统而言，系统的描述都可以采用抽象的数学形式来进行描述。这是因为任何系统都可以被看作是输入到输出的某种变换。例如，离散系统可以由下述的变换进行描述：

$$y(n)=T\{u(n)\},\ n=0,1,2,3\,\mathbf{L}\ \mathbf{L}$$

在离散系统之中，线性离散系统具有重要的地位。下面对线性离散系统进行简单的介绍。在此之前，读者需要理解如下的两个概念：

(1) 齐次性：若对于离散系统，如果对任意的输入与给定的任意常数，恒有

$$T\{au(n)\}=aT\{u(n)\}$$

(2) 叠加性：如果系统对于输入和，输出分别为和，恒有

$$T\{u_1(n)+u_2(n)\}=T\{u_1(n)\}+T\{u_2(n)\}$$

则称系统满足叠加性。

【定义3.3】**线性离散系统**。当离散系统同时满足齐次性与叠加性时，即

$$T\{au_1(n)+bu_2(n)\}=aT\{u_1(n)\}+bT\{u_2(n)\}$$

则称此离散系统为线性离散系统。

例如，对于如下的离散系统：

$$y(n)=u(n)^2+2u(n-1)$$

$$
\begin{aligned}
T\left\{u_1(n)+u_2(n)\right\} &= u_1(n)+u_2(n)+u_1(n-1)+u_2(n-1) \\
&= \{u_1(n)+u_1(n-1)\}+\{u_2(n)+u_2(n-1)\} \\
&= T\left\{u_1(n)\right\}+T\left\{u_2(n)\right\}
\end{aligned}
$$

3.1.1 线性离散系统的数学描述

对于线性离散系统来说，可以使用最一般的方式对其进行描述，如采用如下的数学方程进行描述：

$$y(n) = f\left(u(n) \quad u(n-1)\ \mathbf{L}\ \mathbf{L}\ ; \quad y(n-1) \quad y(n-2)\ \mathbf{L}\ \mathbf{L}\ ;\right)$$

或采用差分方程进行描述：

状态更新方程： $x(n+1) = f_d(x(n), u(n), n)$

系统输出方程： $y(n) = g(x(n), u(n), n)$

除了使用一般的方式描述线性离散系统之外，针对线性离散系统本身的特点，经常使用Z变换来描述线性离散系统。Z变换是对离散信号进行分析的一个强有力的工具，尤其是对线性离散系统。Z变换有丰富的内容，但由于本书的目的主要是对各种实际的系统进行Simulink仿真，故在此仅简单介绍线性离散系统的Z变换域描述以及MATLAB中一些比较常用的对线性离散系统进行分析的函数。

Z变换具有多种不同的性质，这里仅介绍Z变换的如下两个性质：

(1) 线性性。即对于离散信号和，设它们的Z变换分别为与，所谓Z变换的线性性指的是Z变换满足下面的关系：

$$Z\{au_1(n)+bu_2(n)\}=aZ\{u_1(n)\}+bZ\{u_2(n)\}$$

(2) 设离散信号的Z变换为，则的Z变换为。

【例3.3】 对于如下的线性离散系统：

$$y(n) = u(n) + 2u(n-1) + 3y(n-1)$$

同时对等式两边进行Z变换，则有。一般在系统分析中，往往对系统输出与系统输入的比值比较关心，将此式化成分式的形式，有

$$\frac{Y(z)}{U(z)} = \frac{1+2z^{-1}}{1-3z^{-1}} = \frac{z+2}{z-3}$$

$$\frac{Y(z)}{U(z)}=\frac{n_0+n_1z^{-1}+n_2z^{-2}}{d_0+d_1z^{-1}}$$

在对系统进行描述分析时，此种形式的描述称之为滤波器描述。对上式进行等价变换，可以得到系统的传递函数描述线性系统最常用的一种描述方式：

$$\frac{Y(z)}{U(z)}=\frac{n_0z^2+n_1z+n_2}{d_0z^2+d_1z}$$

还可以得到系统的零极点描述：

$$\frac{Y(z)}{U(z)}=k\frac{(z-z_1)(z-z_2)}{z(z-p_1)}$$

3.2.6 线性离散系统的Simulink描述

线性离散系统的描述方式有如下四种形式：

(1) 线性离散系统的滤波器模型：在Simulink中，滤波器表示为num=[n0 n1 n2]; den=[d0 d1]；其中num表示Z变换域分式的分子系数向量，den为分母系数向量。

(2) 线性离散系统的传递函数模型：在Simulink中，系统的传递函数表示为num=[n0 n1 n2]; den=[d0 d1]；

(3) 线性离散系统的零极点模型：在Simulink中，系统零极点表示为gain=K; zeros=[z1, z2]; poles=[0, p1];

(4) 线性离散系统的状态空间模型：在Simulink中，设系统差分方程为如下形式：$x(n+1)=\boldsymbol{F}x(n)+\boldsymbol{G}\boldsymbol{u}(n)$；$\boldsymbol{y}(n)=\boldsymbol{C}x(n)+\boldsymbol{D}\boldsymbol{u}(n)$。其中$x(n)$, $\boldsymbol{u}(n)$, $\boldsymbol{y}(n)$分别为线性离散系统的状态变量、输入向量、输出向量。***F***，***G***，***C***，***D***分别为变换矩阵。在Simulink中，其表示很简单，只需要输入相应的变换矩阵***F***，***G***，***C***，***D***即可。

[3.4] 对于如下的线性离散系统：

$$\frac{Y(z)}{U(z)}=\frac{2z^2-z-5}{z^3+3z^2+6z+2}$$

在MATLAB中输入下面的语句，可以绘制出此系统的Bode图：

```
>>num=[2 –1 –5];
>>den=[1 3 6 2];
>>dbode(num,den,1)
>>grid;
```

此离散系统的Bode图如图3.3所示。

当然也可以用下面的语句求出系统的幅值与相位而不绘制图形：

```
>>[mag,phase]=dbode(num,den,1);
```

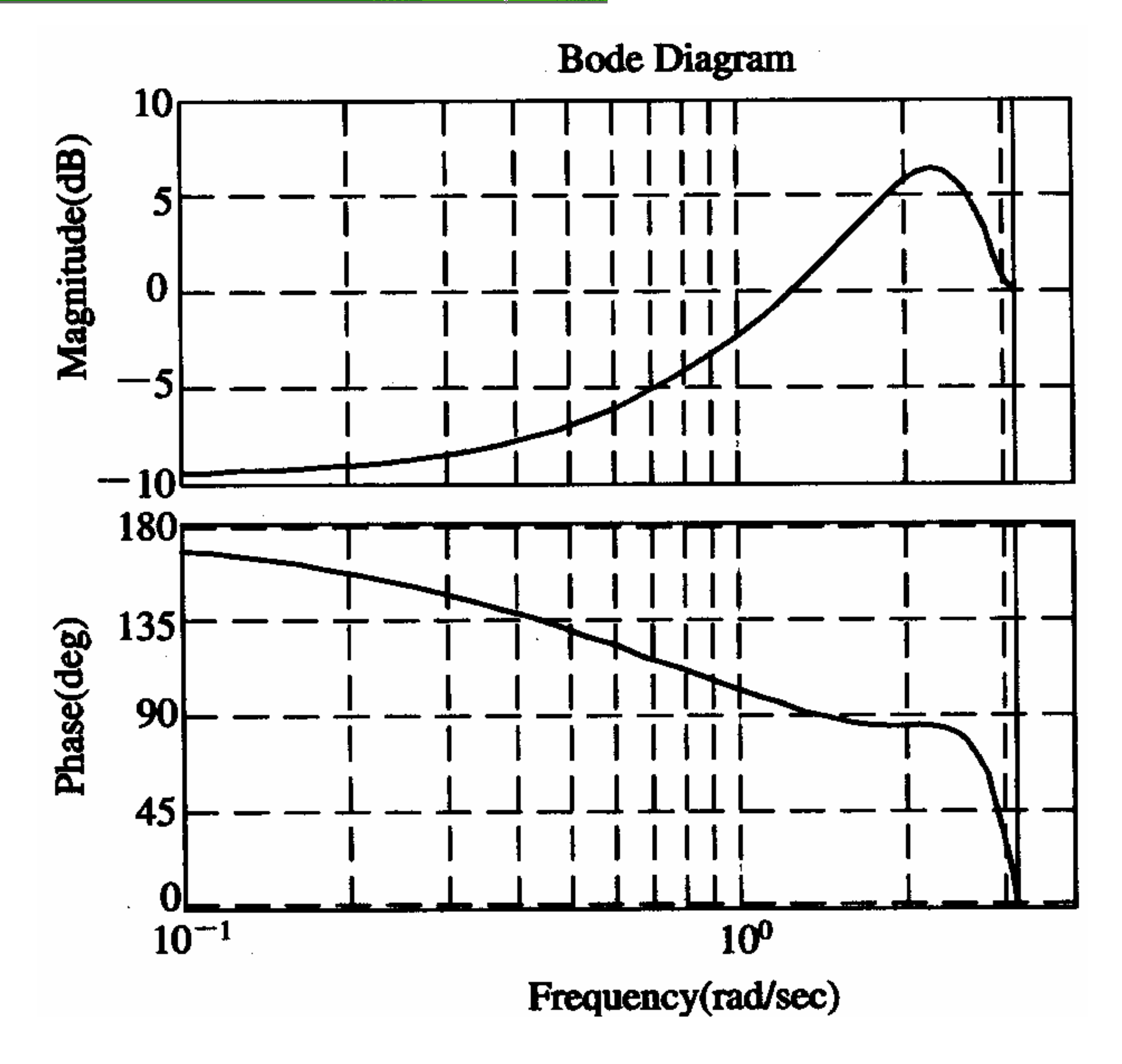


图3.3 线性离散系统的Bode图

此外，在MATLAB中，离散系统的不同描述模型之间可以进行相互转化。这里给出几个比较常用的函数：

[zeros,poles,k]=tf2zp(num,den)　% 将系统传递函数模型转化为零极点模型

[num,den]=zp2tf(zeros,poles,k)　% 将系统零极点模型转化为传递函数模型。其中num，den分别为系统的传递函数表

% 示；zeros，poles，k为系统的零极点模型

至于线性离散系统的状态空间模型描述，这里不再介绍，感兴趣的读者可以参考其它有关的书籍。这里给出它与传递函数模型、零极点模型相互转化的函数命令：

[zeros,poles,k]=ss2zp(F,G,C,D)　　% 将系统状态空间模型转化为零极点模型

[F,G,C,D]=zp2ss(zeros,poles,k)　　% 将系统零极点模型转化为状态空间模型

[num,den]=ss2tf(F,G,C,D)　　% 将系统状态空间模型转化为传递函数模型

[F,G,C,D]=tf2ss(num,den)　　% 将系统传递函数模型转化为状态空间模型

例3.5 以线性离散系统 $\dfrac{Y(z)}{U(z)}=\dfrac{2z^2-z-5}{z^3+3z^2+6z+2}$ 为例说明系统模型的转化。

解： 将传递函数模型转化为零极点模型：

```
>> num=[2  –1  –5];
>>den=[1  3  6  2];
>> [zeros,poles,k]=tf2zp(num,den)
```

结果为

```
>> zeros =
    1.8508
   -1.3508
>>poles =
  -1.2980 + 1.8073i
  -1.2980 - 1.8073i
  -0.4039
>>k =
2.0000
```

将传递函数模型转化为状态空间模型：

```
>> num=[2  -1  -5];
>>den=[1  3  6  2];
>>[F,G,C,D]=tf2ss(num,den)
```

结果为

```
>>F =
   -3.0000    -6.0000    -2.0000
    1.0000          0          0
         0     1.0000          0
```

```
>>G =

    1

    0

    0

>>C =

   2.0000   –1.0000   –5.0000

>>D =

0
```

第5章将对系统仿真作详细的介绍，在此不再赘述。



3.3　连续系统模型及表示

3.1.1 连续系统的基本概念

与离散系统不同，连续系统是指系统输出在时间上连续变化，而非仅在离散的时刻采样取值。连续系统的应用非常广泛，下面给出连续系统的基本概念。

【定义3.4】　**连续系统**。满足如下条件的系统为连续系统：

(1) 系统输出连续变化。变化的间隔为无穷小量。

(2) 对系统的数学描述来说，存在系统输入或输出的微分项（导数项）。

(3) 系统具有连续的状态。在离散系统中，系统的状态为时间的离散函数，而连续系统的状态为时间连续量。

3.1.1 连续系统的数学描述

设连续系统的输入变量为，其中为连续取值的时间变量，设系统的输出为；由连续系统的基本概念可以写出连续系统的最一般的数学描述，即

$$y(t) = f_c(u(t), t)$$

系统的实质为输入变量到输出变量的变换，注意这里系统的输入变量与输出变量既可以是标量（单输入单输出系统），也可以是向量（多输入多输出系统）；而且在系统的数学描述中含有系统输入或输出的导数。

除了采用最一般的数学方程描述连续系统外，还可以使用连续系统的微分方程形式对连续系统进行描述，即

$$\dot{x}(t) = f_c(x(t), u(t), t) \rightarrow \text{微分方程}$$

$$y(t) = g(x(t), u(t), t) \rightarrow \text{输出方程}$$

这里分别为连续系统的状态变量、状态变量的微分。对于线性连续系统来说，由连续系统的微分方程描述可以容易地推导出连续系统的状态空间模型。这与使用差分方程对离散系统进行描述相类似。下面举例说明连续系统的数学描述。

【例3.6】 对于如下的连续系统：

$$y(t) = u(t) + \dot{u}(t) \qquad u(t) = t + \sin t, \quad t \geqslant 0$$

显然此系统为单输入单输出连续系统，且含有输入变量的微分项。由此方程可以很容易得出系统的输出变量为

$$y(t) = t + \sin t + 1 + \cos t = t + \sin t + \cos t + 1, \quad t \geqslant 0$$

3.3.3 连续系统的Simulink描述

前面给出了连续系统的基本概念与系统的基本描述方法：数学方程描述与微分方程描述。本部分使用【例3.6】给出的连续系统

$$y(t) = u(t) + \dot{u}(t) \qquad u(t) = t + \sin t,$$

说明如何利用Simulink对连续系统进行描述，并在此基础上对连续系统进行简单分析。与前面类似，在此并不建立系统的Simulink模型进行仿真，而是采用编写M脚本文件从原理上对连续系统进行说明，并进行简单的仿真。

【例3.7】 编写脚本文件systemdemo3.m，对【例3.6】中的连续系统进行分析。

```
% systemdemo3.m脚本文件
t=0:0.1:5;                  %系统仿真范围，时间间隔为0.1 s
ut=t+sin(t);                %系统输入变量
utdot=1+cos(t);             %系统输入变量的导数
yt=ut+utdot;                %系统输出
plot(yt);grid;   %绘制系统输出曲线
```

图3.4为此连续系统在时间[0, 5]内的输出曲线。由此可见，使用简单的MATLAB语句可对系统性能进行简单的分析。

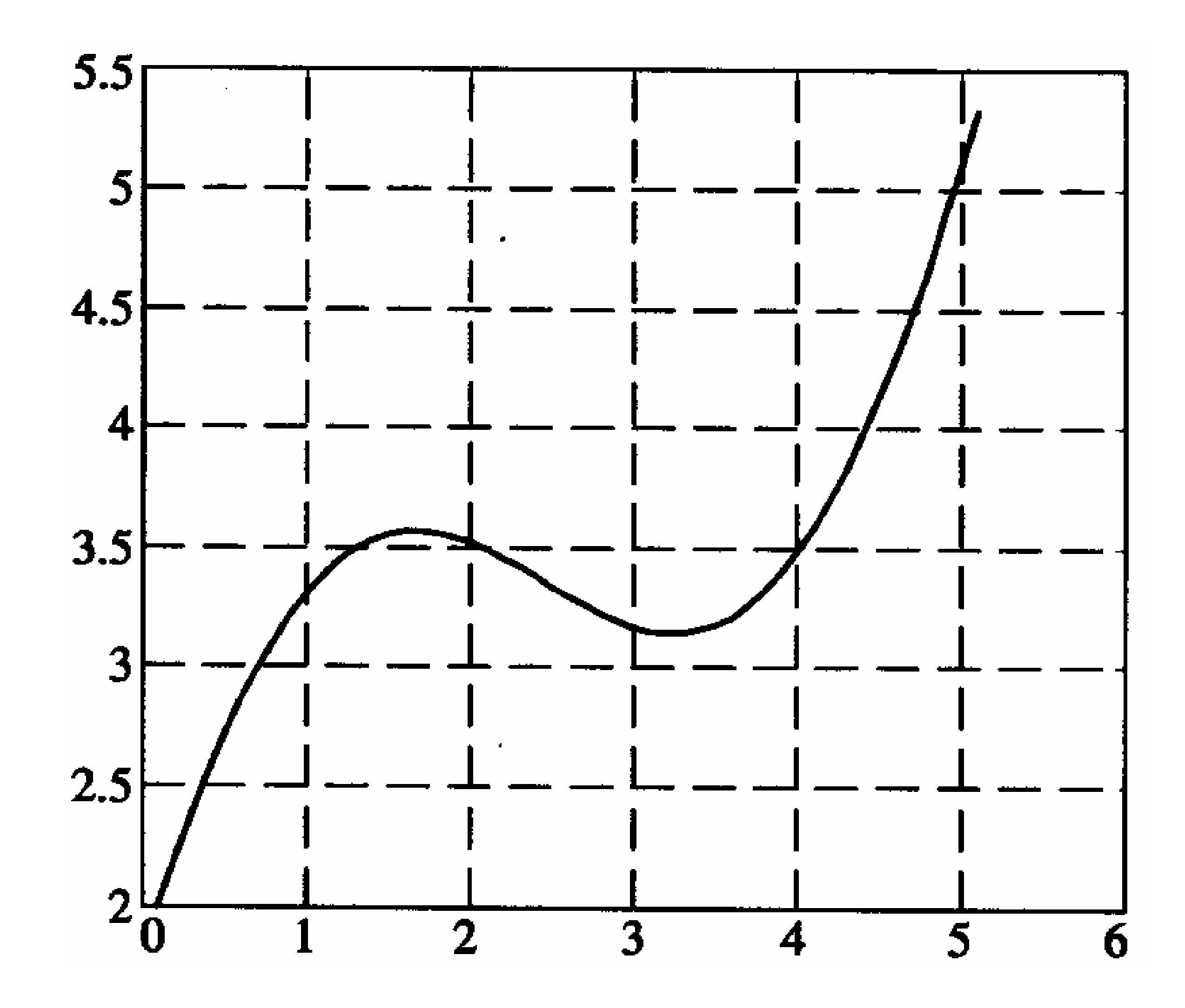


图3.4　连续系统输入输出关系图

3.3.4 线性连续系统

在介绍线性离散系统时，已经给出线性系统的基本概念，这里做一个简单的回顾并介绍线性连续系统的概念。连续系统可以用如下的方式来表达：

$$y(t)=T\{u(t)\}$$

【定义3.5】 **线性连续系统**。如果一个连续系统能够同时满足如下的性质：

(1) 齐次性。对于任意的参数，系统满足

$$T\{au(t)\}=aT\{u(t)\}$$

(2) 叠加性。对于任意输入变量与，系统满足

$$T\{u_1(t)+u_2(t)\}=T\{u_1(t)\}+T\{u_2(t)\}$$

则此连续系统为线性连续系统。

下面举例说明。如对【例3.6】中的连续系统：

$$y(t)=u(t)+\dot{u}(t) \qquad u(t)=t+\sin t, \qquad t\geqslant 0$$

3.3.5 线性连续系统的数学描述

线性连续系统最一般的描述为连续系统的输入输出方程形式，即，也可以使用连续系统的微分方程模型进行描述：

$$\dot{x}(t) = f_c(x(t),u(t),t) \rightarrow \text{微分方程}$$
$$y(t) = g(x(t),u(t),t) \rightarrow \text{输出方程}$$

除了使用这两种连续系统通用的形式描述线性连续系统之外，还可以使用传递函数、零极点模型与状态空间模型对其进行描述。与线性离散系统相类似，线性连续系统的传递函数模型与零极点模型采用连续信号的拉氏变换来实现。

拉氏变换具有如下两个性质：

(1) 线性性。即对于连续信号和，设它们的拉氏变换分别为与，则拉氏变换的线性性是指拉氏变换满足下面的关系：

$$L\{au_1(t)+bu_2(t)\}=aL\{u_1(t)\}+bL\{u_2(t)\}$$

(2) 设连续信号的 $u(t)$ 拉氏变换为 $U(s)$，则 $\dot{u}(t)$ 的拉氏变换为 $sU(s)$，$\ddot{u}(t)$ 的拉氏变换为 $s^2U(s)$。

$$u(t)=m\ddot{y}(t)+b\dot{y}(t)+ky(t) \qquad m\neq 0$$

同时对等式的两边进行拉氏变换，则有。将其化为分式的形式，则有

$$\frac{Y(s)}{U(s)} = \frac{1}{ms^2 + bs + k}$$

这便是系统的传递函数模型。

一般来说，线性连续系统的拉氏变换总可以写成如下传递函数的形式：

$$\frac{Y(s)}{U(s)} = \frac{n_0 s + n_1}{d_0 s^2 + d_1 s + d_2}$$

将其进行一定的等价变换，可以得出线性连续系统的零极点模型：

$$\frac{Y(s)}{U(s)} = k\frac{s-z_1}{(s-p_1)(s-p_2)}$$

其中为线性连续系统的零点，、为系统的极点，为系统的增益。

线性连续系统的另外一种模型为状态空间模型。前面已经提到，对于线性连续系统，使用其微分方程很容易推导出系统的状态空间模型。这里给出线性连续系统用状态空间模型进行描述的一般方式：

$$\begin{cases} \dot{x}(t) = Ax(t) + Bu(t) \\ y(t) = Cx(t) + Du(t) \end{cases}$$

其中，$x(t)$ 为线性连续系统的状态变量,$u(t)$、$y(t)$ 分别为系统的输入与输出变量，可以为标量，也可以为向量。下面介绍如何在Simulink中实现对线性连续系统的描述。

3.1.1 线性连续系统的Simulink描述

一般来说，在Simulink中对线性连续系统的描述方式有以下三种：

(1) 线性连续系统的传递函数模型描述：在Simulink中，传递函数表示为num=[n0,n1]; den=[d0,d1,d2]; 其中num表示传递函数的分子系数向量，den为分母系数向量。

(2) 线性连续系统的零极点模型描述：在Simulink中，零极点模型表示为gain=k; zeros=z1; poles=[p1,p2]; 其中gain表示系统增益，zeros表示系统零点，poles表示系统极点。

(3) 线性连续系统的状态空间模型描述：如果系统的状态空间表示为

$$\begin{cases} \dot{\boldsymbol{x}}(t) = \boldsymbol{A}x(t) + \boldsymbol{B}u(t) \\ \boldsymbol{y}(t) = \boldsymbol{C}x(t) + \boldsymbol{D}u(t) \end{cases}$$

则在Simulink中直接输入变换矩阵***A***，***B***，***C***，***D***即可。

一般来说，线性连续系统的不同模型之间可以相互转化，MATLAB中有内置的函数可以完成线性连续系统模型间的转化。我们在线性离散系统模型间转化中已经做了介绍，这里仅列出这些函数原型：

```
[zeros,poles,k]=tf2zp(num,den);
[num,den]=zp2tf(zeros,poles,k);
[zeros,poles,k]=ss2zp(A,B,C,D);
[A,B,C,D]=zp2ss(zeros,poles,k)
[num,den]=ss2tfA,B,C,D)
[A,B,C,D]=tf2ss(num,den)
```

【例3.8】 对于如下采用传递函数模型进行描述的线性连续系统：

$$\frac{Y(s)}{U(s)}=\frac{s-3}{2s^2-3s-5}$$

要求绘制此系统的Bode图、Nyquist图，并求取系统的零极点模型与状态空间模型描述。

解：在MATLAB中输入下面的语句即可：

```
>>num=[1, -3];
>>den=[2, -3, -5];
>>w=logspace(-1, 1);
```

```
>>subplot(2,1,1); bode(num, den, w);
   >>subplot(2,1,2); nyquist(num,den,w);
>>[zeros, poles, k]=tf2zp(num,den)
>>[A,B,C,D]=tf2ss(num,den)
```

系统的Bode图与Nyquist图如图3.5所示。

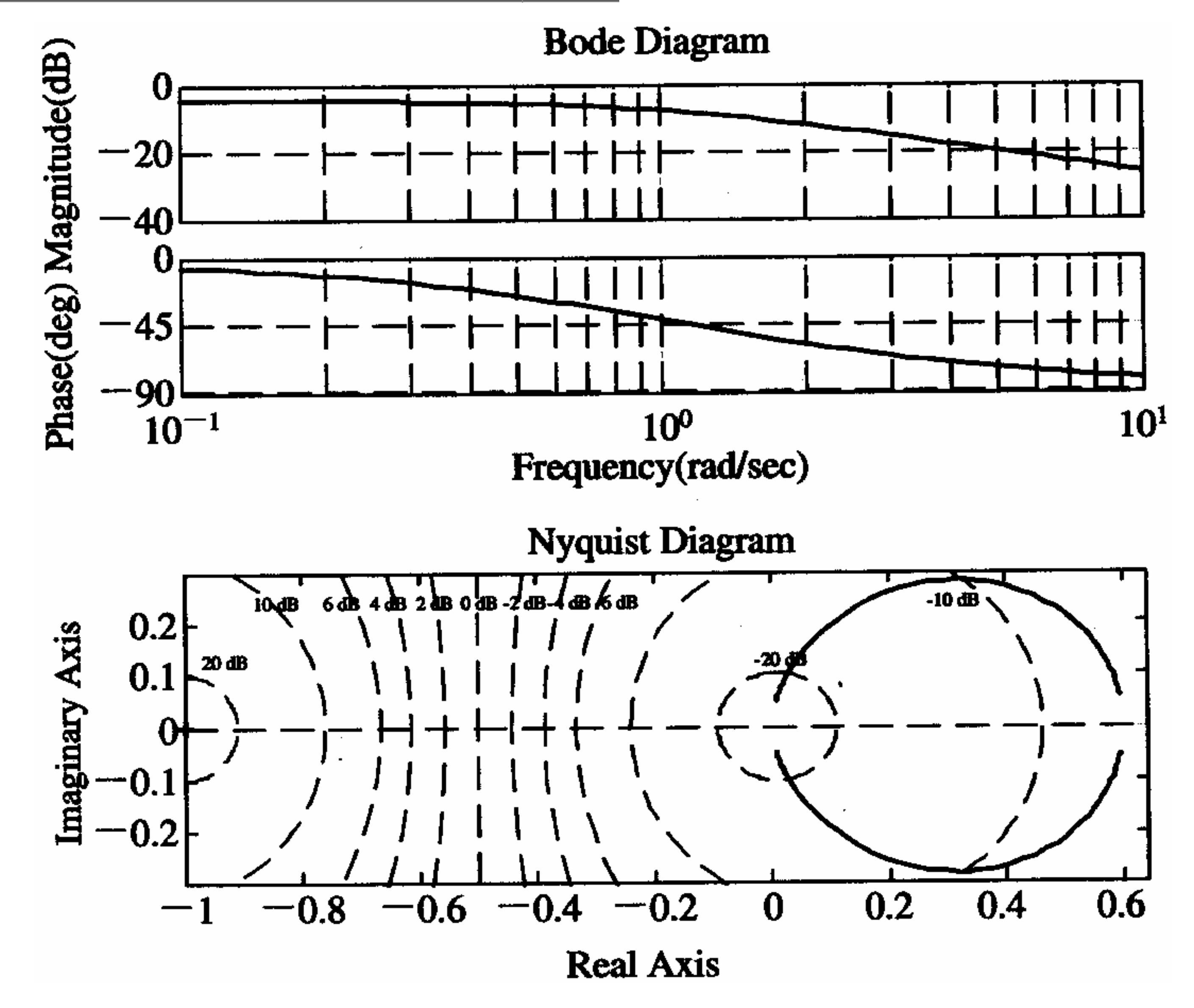


图3.5 线性连续系统的Bode图与Nyquist图

系统的零极点模型与状态空间模型如下所示：

```
>>zeros =
     3
poles =
    2.5000
   -1.0000
k =
    0.5000
A =
    1.5000    2.5000
    1.0000    0
```

```
B =

     1

     0

C =

    0.5000   -1.5000

D =

     0
```

3.4 混合系统模型及表示

3.4.1 混合系统的数学描述

混合系统是由不同类型的系统共同构成的，因此混合系统的数学描述可以由不同类型系统描述共同构成。但是由于混合系统的复杂性，一般难以用单独的数学模型进行描述或表达，因此混合系统一般都是由系统各部分输入与输出间的数学方程所共同描述的，下面举例说明。

【例3.9】 对于如下的一个混合系统：设系统的输入为一离散变量，系统由离散系统与连续系统串联构成，其中离散系统输出经过一个零阶保持器后作为连续系统的输入。其中离散系统的输入输出方程为且，系统采样时间为T_s=1 s。

连续系统的输入输出方程为

$$y(t)=\sqrt{u(t)}+\sin u(t)$$

由于此混合系统中离散系统的输出经过一零阶保持器后作为连续系统的输入，因此与的数学关系为

$$u(t)=y(n),\quad nT_s \quad t<(n+1)T_s$$

其中T_s=1s为离散系统的采样时间。故此混合系统的输入与输出之间的关系可以由下面的方程来描述：

$$\begin{cases} u(n) = n/2, & n = 1,2,3\cdots\cdots \\ y(n) = u(n) + 1, & \\ y(t) = \sqrt{y(n)} + \sin(y(n)), & n \le t < n+1 \end{cases}$$

3.4.2 混合系统的Simulink描述与简单分析

在对单独离散系统或连续系统进行描述时，由于系统一般比较简单，因而可以采用诸如差分方程、传递函数、状态空间等模型表示。但对于混合系统，由于系统本身的复杂性，即使是很简单的混合系统，如【例3.9】给出的例子，都难以用一个简单的模型进行描述。因此，这里采用简单的数学方式对系统进行描述与分析。

【例3.10】 编写M脚本文件systemdemo4.m，对【例3.9】中的混合系统进行分析。

```
%systemdemo4.m文件
t=1:0.1:99.9;   %表示在时间[1，99.9]范围内分析系统。时间间隔0.1 s
n=1:100;                %表示系统输入时刻为1~100 s
un=0.5*n;               %表示系统输入u(n)
yn=un+1;                %表示系统中离散部分的输出，即连续部分的输入
for i=1:length(n)-1
  for j=1:length(t)
```

```
if t(j)>=n(i)&t(j)<n(i+1)                    %判断连续部分的输入时间范围
        y(j)=sqrt(yn(n(i)))+sin(yn(n(i)));     %计算系统输出
    end
  end
end
plot(t,y);grid;              %绘制系统输出曲线图系统输出曲线图如图3.6所示。
```

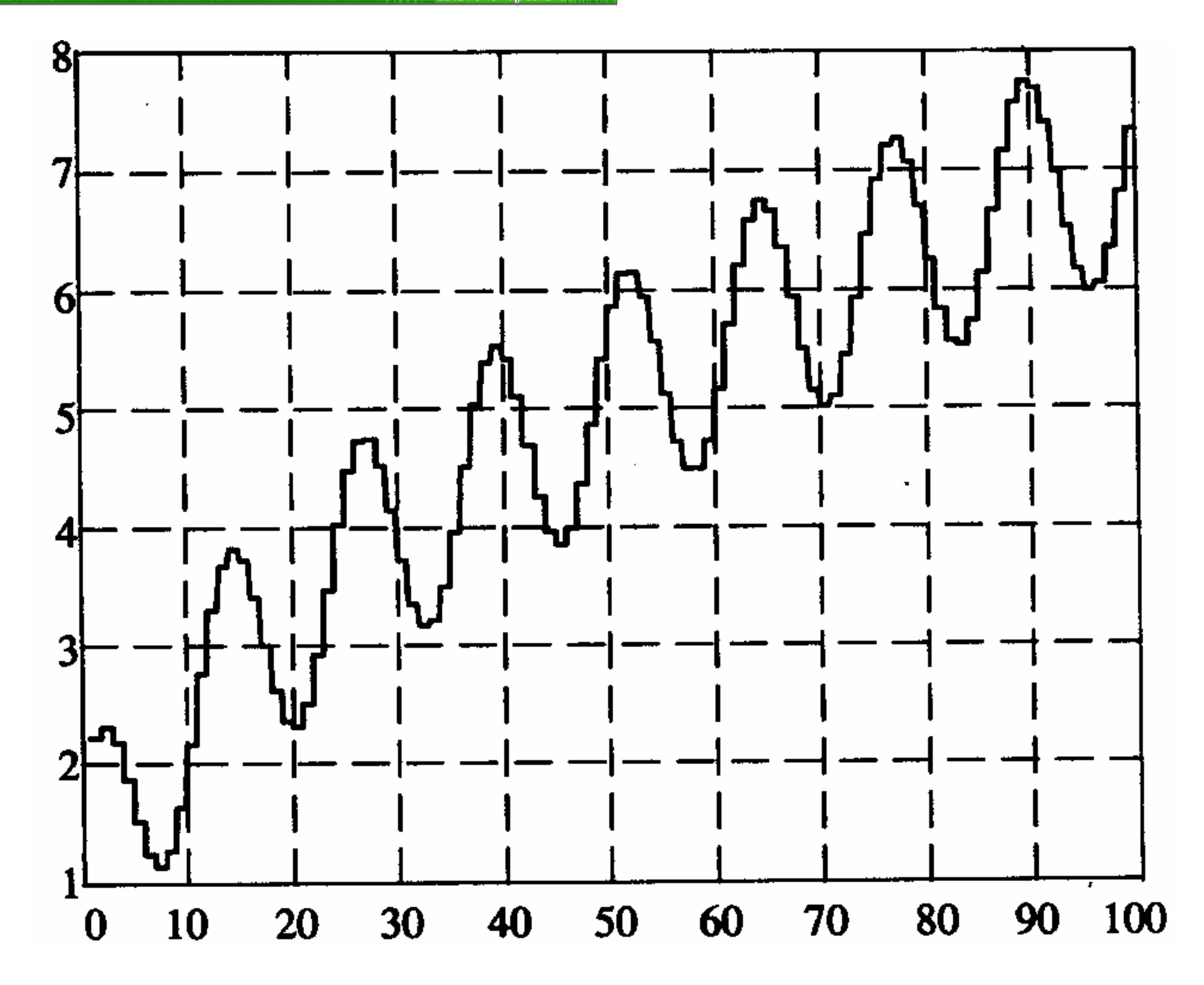


图3.6 混合系统输入输出关系图

从系统输出曲线图3.6中可以看出：由于系统中离散部分的输出经过零阶保持器后作为连续部分的输入，而零阶保持器具有阶跃的特性，在系统仿真结果中出现阶跃现象。另外，系统呈现类似正弦发散的特征表明系统为一发散不稳定系统。

本章对动态系统做了简单的介绍，其中涉及到简单系统、离散系统、线性离散系统、连续系统、线性连续系统、混合系统等系统的概念以及数学描述和Simulink描述，并且使用简单的MATLAB语句对不同的系统进行了简单的仿真与分析。至于使用Simulink进行动态系统建模、仿真与分析将在第二部分进行详细介绍。



第4章　创建Simulink模型



4.1 启用Simulink并建立系统模型

由于Simulink是基于MATLAB环境之上的高性能的系统级仿真设计平台，因此启动Simulink之前必须首先运行MATLAB，然后才能启动Simulink并建立系统模型。启动Simulink有两种方式：

(1) 用命令行方式启动Simulink。即在MATLAB的命令窗口中直接键入如下命令：

```
>>simulink
```

(2) 使用工具栏按钮启动Simulink。即用鼠标单击MATLAB工具栏中的Simulink按钮。

启动Simulink，建立系统模型，其相应的基本操作如图4.1所示。

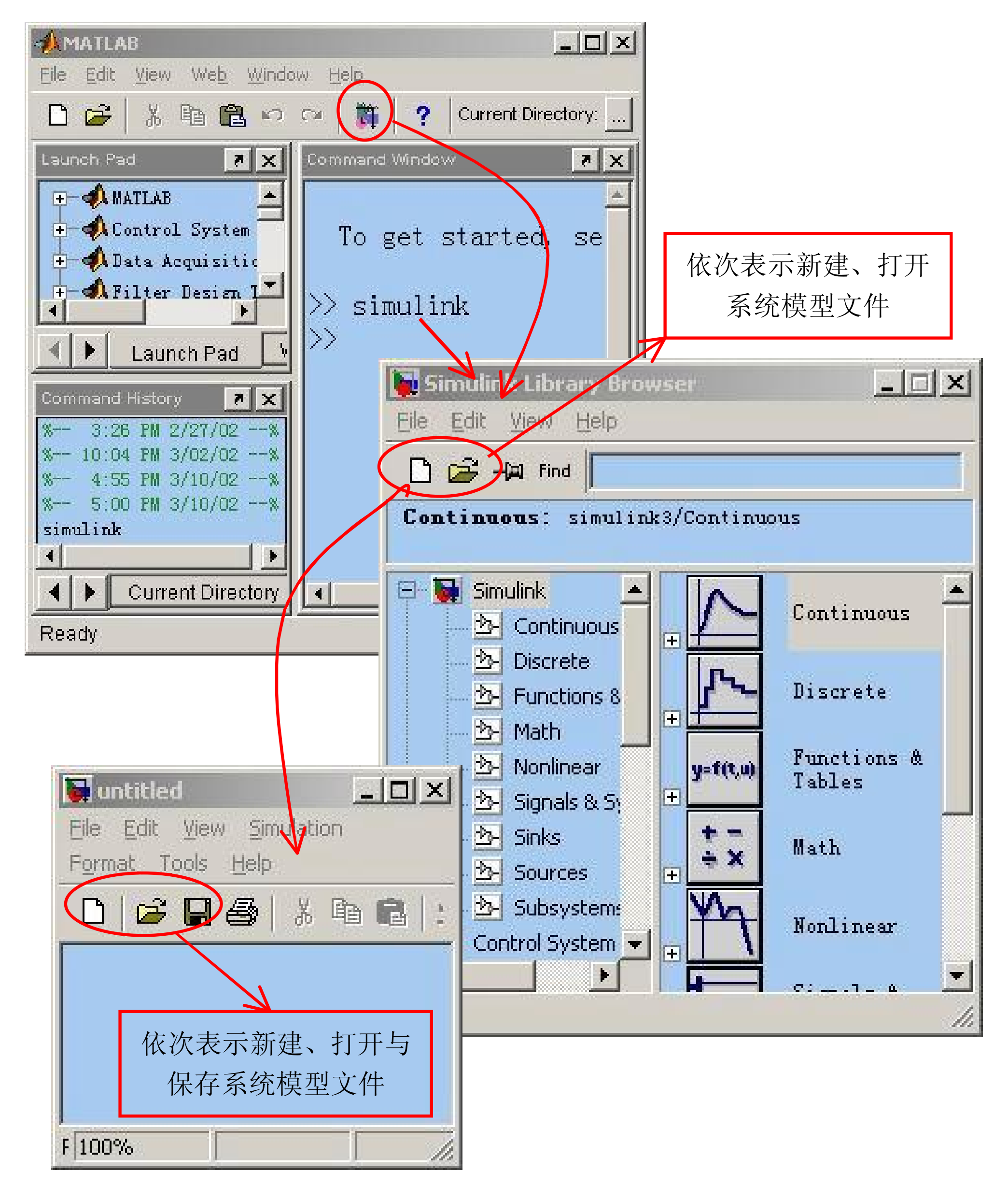


图4.1 启动Simulink，建立系统模型的基本操作

当用户完成Simulink系统模型的编辑之后，需要保存系统模型，然后设置模块参数与系统仿真参数，最后便可以进行系统的仿真。

无论采用何种方式，用户都可以在短短几分钟内熟练掌握启动Simulink的方法并开始创建动态系统模型。在系统模型编辑器中，用户可以“拖动”Simulink提供的大量的内置模块建立系统模型。下一节将对Simulink中的内置系统模块作一个比较全面的介绍，以便初学者无需查阅各个模块的帮助文献，便可以迅速建立所需的系统模型。



4.2 Simulink模块库简介与使用

在4.1节中，用户已经掌握了如何启动Simulink并新建一个动态系统模型。为便于用户能够快速构建自己所需的动态系统，Simulink提供了大量以图形方式给出的内置系统模块，使用这些内置模块可以快速方便地设计出特定的动态系统。为了便于用户对Simulink内置模块库的认识与使用，本节简单介绍Simulink中的模块库以及模块库中具有代表意义的系统模块。图4.2所示为Simulink的模块库浏览器。



图4.2 Simulink的模块库浏览器

Simulink的模块库能够对系统模块进行有效的管理与组织，使用Simulink模块库浏览器可以按照类型选择合适的系统模块、获得系统模块的简单描述以及查找系统模块等，并且可以直接将模块库中的模块拖动或者拷贝到用户的系统模型中以构建动态系统模型。

4.2.1 Simulink公共模块库

Simulink公共模块库是Simulink中最为基础、最为通用的模块库，它可以被应用到不同的专业领域中。Simulink公共模块库共包含9个模块库，如图4.3所示。下面分别介绍各个模块的功能：

1. Continuous（连续系统模块库）

连续系统模块库以及其中各模块的功能如图4.4所示。

2. Discrete（离散系统模块库）

离散系统模块库以及其中各模块的功能如图4.5所示。

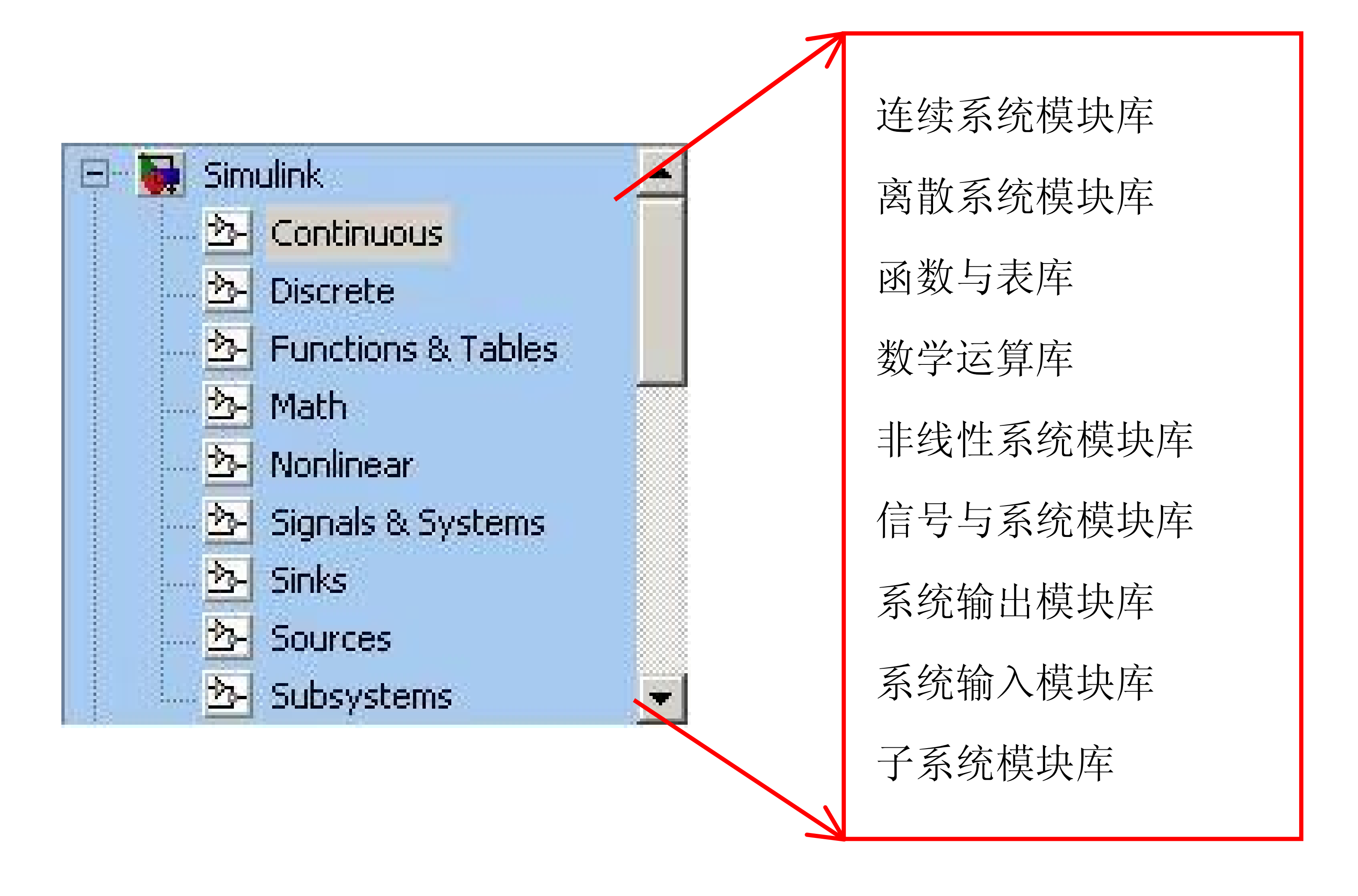


图4.3 Simulink的公共模块库

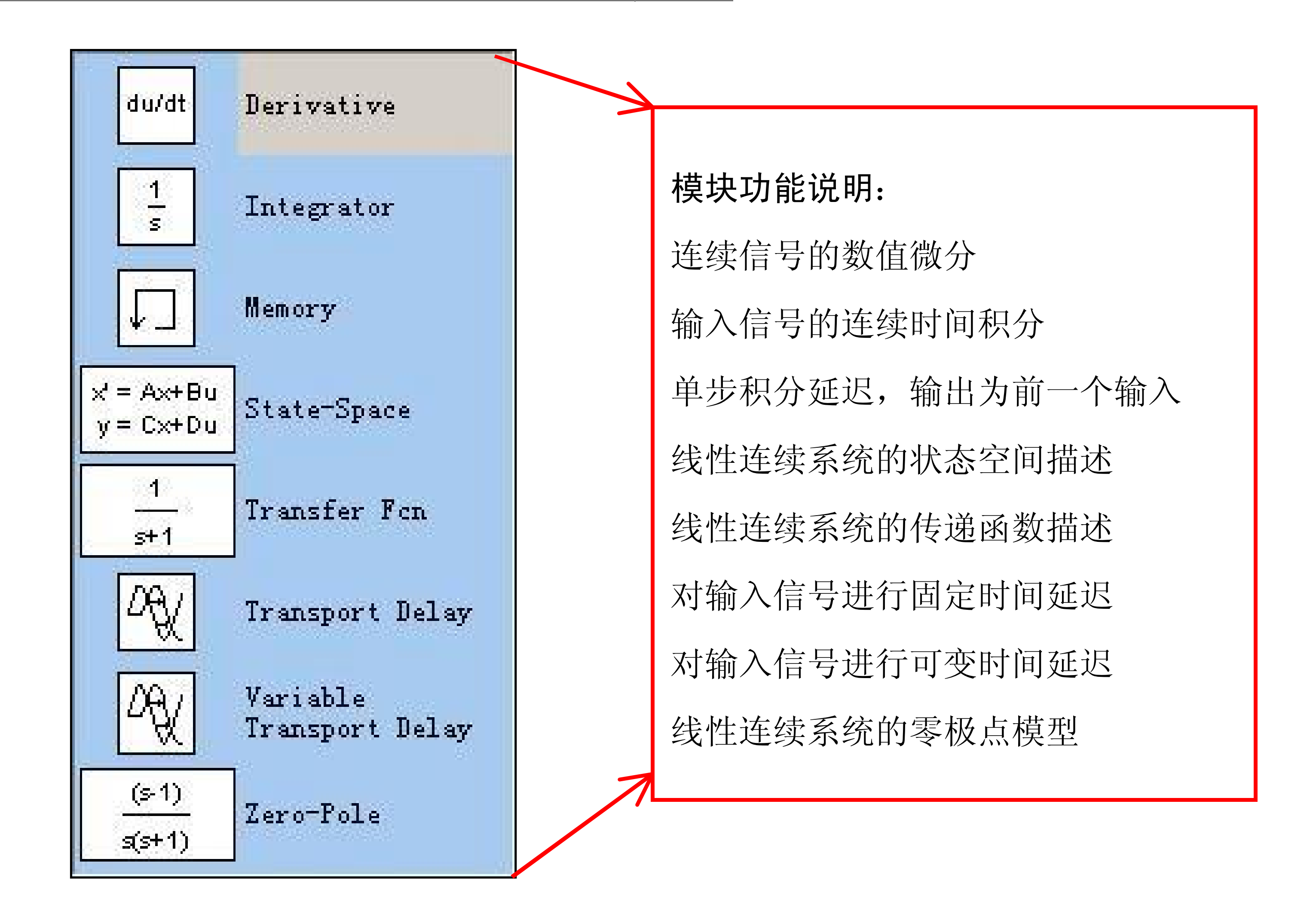


图4.4 连续系统模块库及其功能

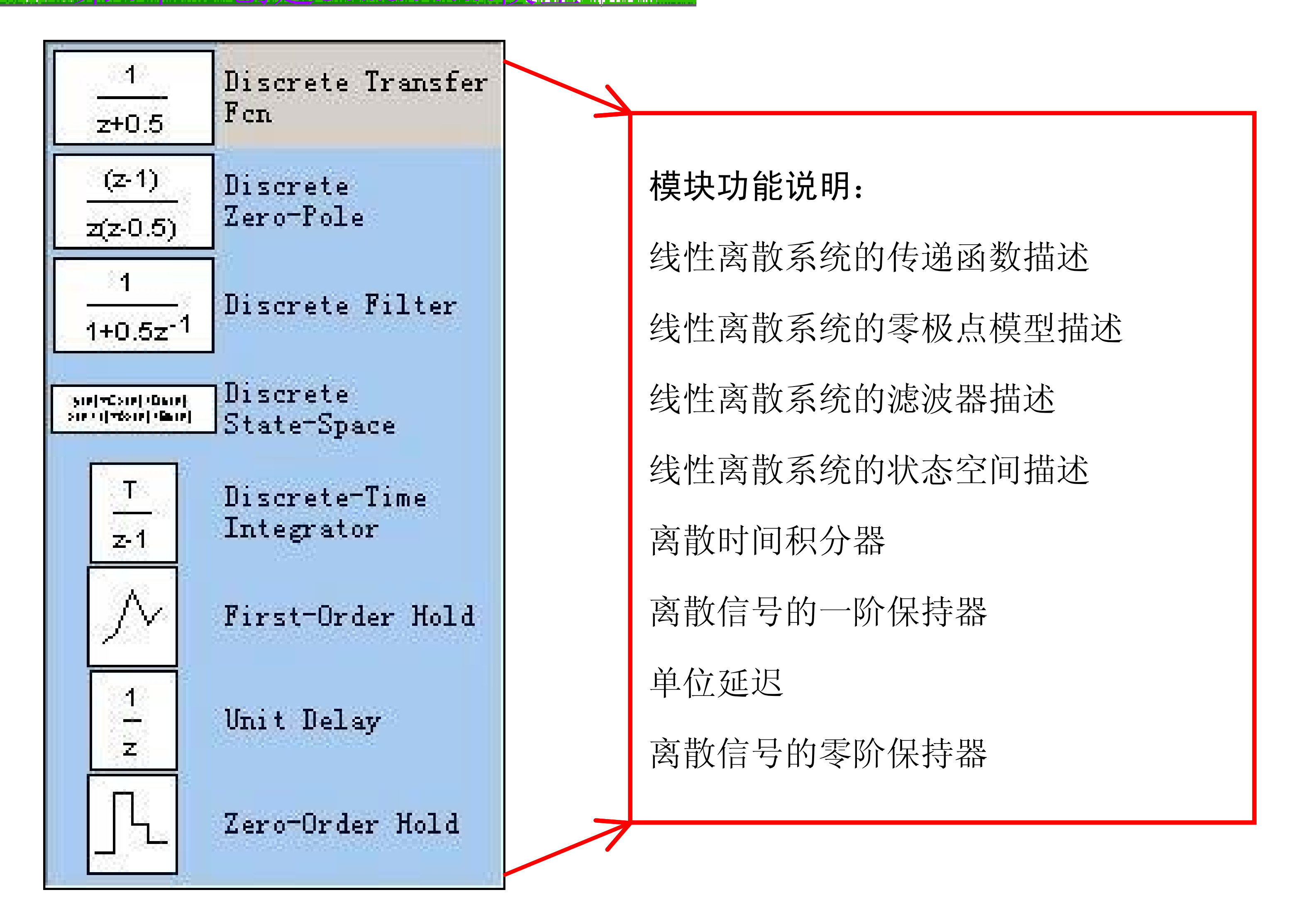


图4.5 离散系统模块库及其功能

3. Functions & Tables（函数与表库）

函数与表库以及其中各模块的功能如图4.6所示。

4. Math（数学运算库）

数学运算库以及其中各模块的功能如图4.7所示。

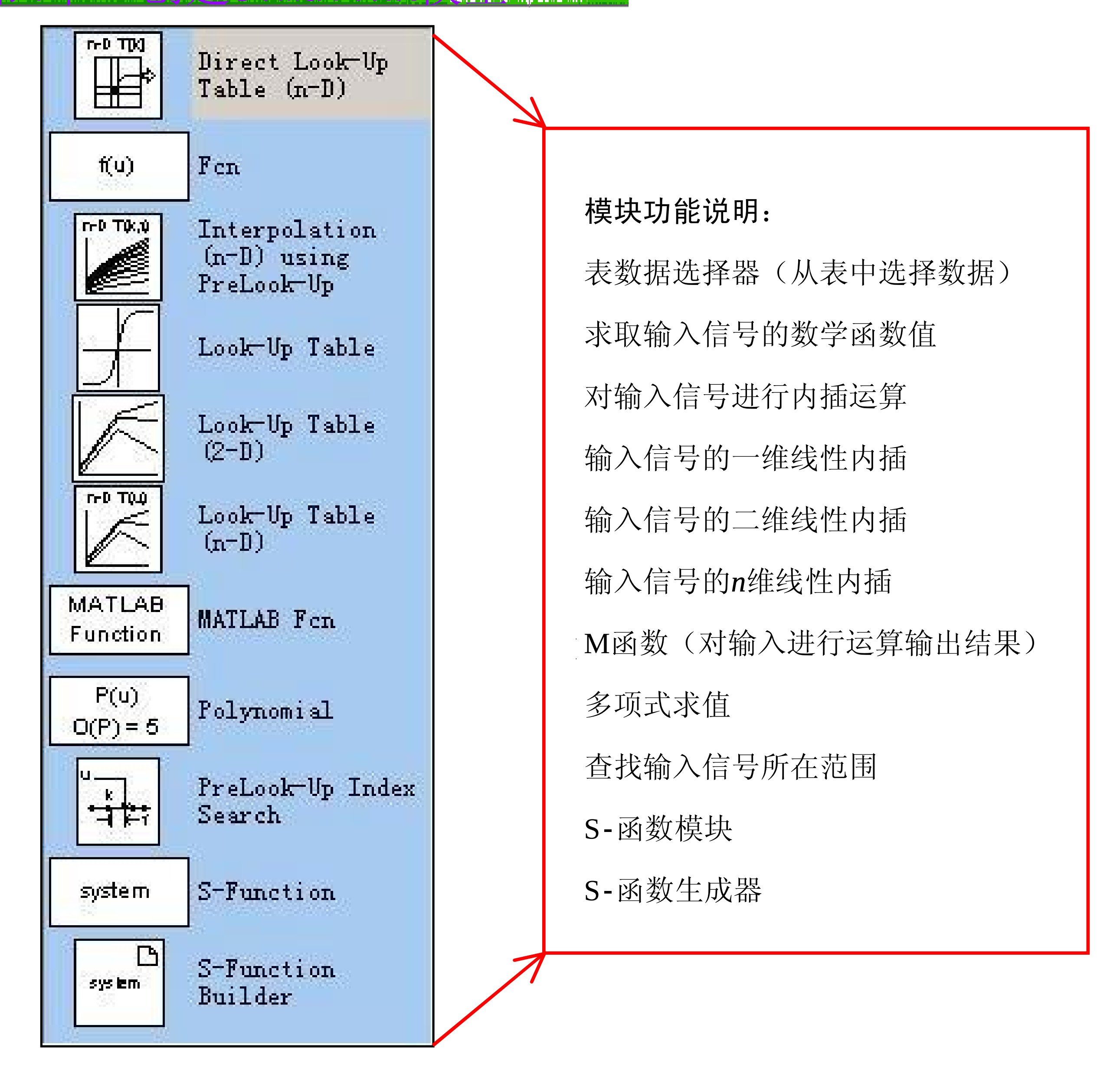


图4.6 函数与表库及其功能



模块功能说明：

求取信号的绝对值

输出强制系统输入为零的代数状态

按位逻辑运算

逻辑真值查找

输出输入复数的幅值与相位

输出系统输入的实部或虚部

点乘运算

信号增益

信号逻辑运算

幅值与相位转化为复数形式

特定的一些数学函数

矩阵增益

求取输入的最小或最大值

乘法或除法器

从输入实部与虚部构造复数

关系运算器

求整运算器

符号运算

渐变增益

对输入求和或差

三角与双曲函数

图4.7 数学运算库及其能

5. Nonlinear（非线性系统模块库）

非线性系统模块库以及其中各模块的功能如图4.8所示。

6. Signals & Systems（信号与系统模块库）

信号与系统模块库以及其中各模块的功能如图4.9所示。



图4.8 非线性系统模块库及其功能



模块功能说明：

数据类型转换器

信号分解器

从Goto模块中获得信号

函数调用发生器

向Goto模块传递信号

Goto模块标记控制器

将信号与特定的偏移值比较

初始化信号

矩阵串联器

合并输入信号为一个输出

模块控制信息

信号组合器

信号探测器

信号维数改变器

选择或重组信号

信号线属性修改

输入信号宽度



图4.9 信号与系统模块库及其功能

7. Sinks（系统输出模块库）

系统输出模块库以及其中各模块的功能如图4.10所示。

8. Sources（系统输入模块库）

系统输入模块库以及其中各模块的功能如图4.11所示。

9. Subsystems（子系统模块库）

子系统模块库以及其中各模块功能如图4.12所示。

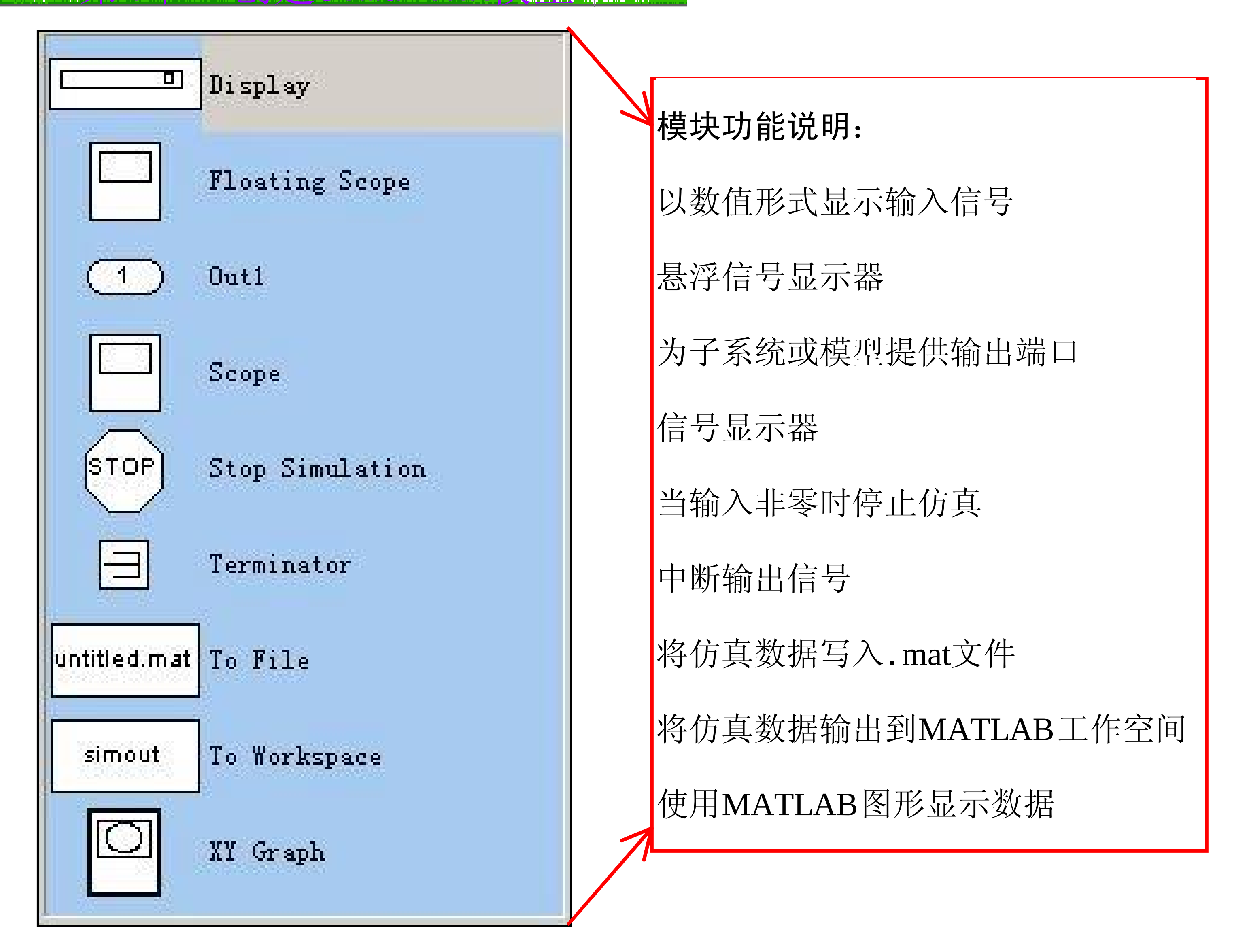


图4.10 系统输出模块库及其功能



图4.11 系统输入模块库及其功能



模块功能说明：

可配置子系统

原子子系统

使能子系统

使能触发子系统

For循环子系统

函数调用子系统

If条件子系统

条件执行子系统

通用子系统

Switch-Case子系统

Switch-Case动作子系统

触发子系统

当型循环子系统

图4.12 子系统模块库及其功能

之所以用较多的篇幅对Simulink的公共模块库进行比较全面的介绍，是因为Simulink的公共模块库中提供了大量内置的系统模块，这些系统模块的用途非常广泛，并且一般的动态系统模型都可以使用公共模块库中的模块来构建。

除了公共模块库之外，Simulink中还集成了许多面向不同专业领域的专业模块库，普通用户一般很少用到其中的模块。因此，在介绍Simulink的专业模块库时，仅对模块库的总体功能做简单的概述。如果用户需要的话，可以在Simulink中的模块描述栏了解其主要功能。

4.1.1 Simulink专业模块库

Simulink集成了许多面向各专业领域的系统模块库，不同领域的系统设计者可以使用这些系统模块快速构建自己的系统模型，然后在此基础上进行系统的仿真与分析，从而完成系统设计的任务。这里仅简单介绍部分专业模块库的主要功能。

(1) Control System Toolbox模块库：面向控制系统的设计与分析，主要提供线性时不变系统的模块。

(2)　DSP Blockset模块库：面向数字信号处理系统的设计与分析，主要提供DSP输入模块、DSP输出模块、信号预测与估计模块、滤波器模块、DSP数学函数库、量化器模块、信号管理模块、信号操作模块、统计模块以及信号变换模块等。

(3)　Simulink Extras模块库：主要补充Simulink公共模块库，提供附加连续模块库、附加线性系统模块库、附加输出模块库、触发器模块库、线性化模块库、系统转换模块库以及航空航天系统模块库等。

(4) S-function demos模块库：主要提供C++、C、FORTRAN以及M文件下S-函数的模块库的演示模块。

(5) Real-Time Workshop与Real-Time Windows Target模块库：主要提供各种用来进行独立可执行代码或嵌入式代码生成，以实现高效实时仿真的模块。它们和RTW、TLC有着密切的联系。

(6) Stateflow库：对使用状态图所表达的有限状态机模型进行建模仿真和代码生成。有限状态机用来描述基于事件的控制逻辑，也可用于描述响应型系统。

(7) 定点模块库：包含一组用于定点算法仿真的模块。

(8) 通讯模块库：专用于通信系统仿真的一组模块。

(9)　Dials & Gauges库：图形仪表模块库，它们实际上是一组ActiveX控件。

(10) 神经网络模块库：用于神经网络的分析设计和实现的一组模块。

(11) 模糊控制模块库：包括一组有关模糊控制的分析设计和实现的模块。

(12)　xPC模块库：提供了一组用于xPC仿真的模块。



4.3 构建Simulink框图

4.2节中简单介绍了Simulink中的一些比较常用的系统模块。本节将介绍如何使用这些系统模块以构建用户自己的系统模型。当Simulink库浏览器被启动之后，通过鼠标左键单击模块库的名称可以查看模块库中的模块。模块库中包含的系统模块显示在Simulink库浏览器右边的一栏中。对Simulink库浏览器的基本操作有：

(1) 使用鼠标左键单击系统模块库，如果模块库为多层结构，则单击“+”号载入库。

(2) 使用鼠标右键单击系统模块库，在单独的窗口打开库。

(3) 使用鼠标左键单击系统模块，在模块描述栏中显示此模块的描述。

(4) 使用鼠标右键单击系统模块，可以得到系统模块的帮助信息，将系统模块插入到系统模型中，查看系统模块的参数设置，以及回到系统模块的上一层库。

此外还可以进行以下操作：

(1) 使用鼠标左键选择并拖动系统模块，并将其拷贝到系统模型中。

(2) 在模块搜索栏中搜索所需的系统模块。

4.3.1 模块选择

这里用一个非常简单的例子介绍如何建立动态系统模型。此简单系统的输入为一个正弦波信号，输出为此正弦波信号与一个常数的乘积。要求建立系统模型，并以图形方式输出系统运算结果。已知系统的数学描述为

系统输入： $u(t) = \sin t, t \geqslant 0$

系统输出： $y(t) = au(t), a \neq 0$

启动Simulink并新建一个系统模型文件。欲建立此简单系统的模型，需要如下的系统模块（均在Simulink公共模块库中）：

(1) 系统输入模块库Sources中的Sine Wave模块：产生一个正弦波信号。

(2) 数学库Math中的Gain模块：将信号乘上一个常数（即信号增益）。

(3) 系统输出库Sinks中的Scope模块：图形方式显示结果。

选择相应的系统模块并将其拷贝（或拖动）到新建的系统模型中，如图4.13所示。

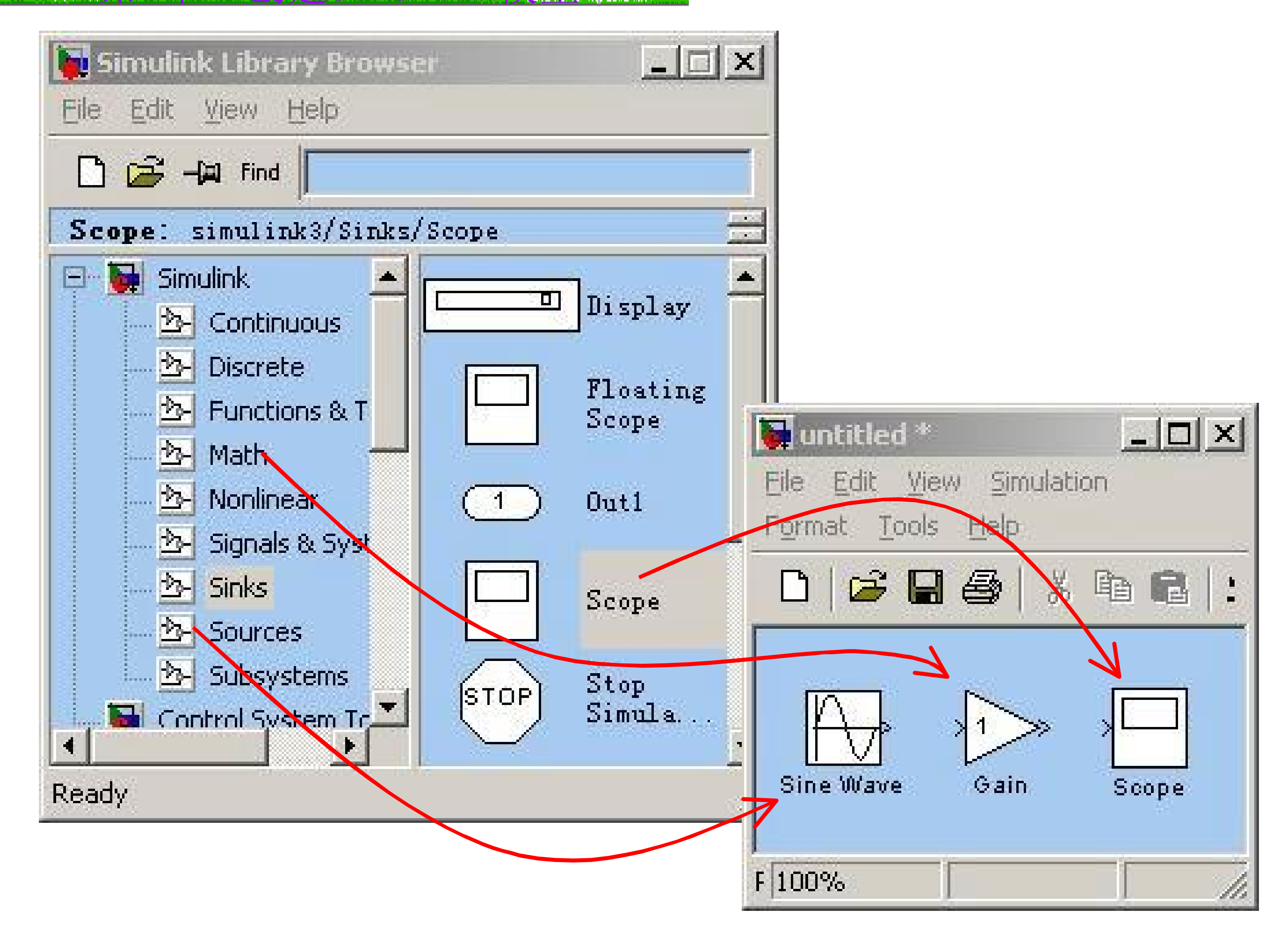


图4.13 选择系统所需模块

在选择构建系统模型所需的所有模块后，需要按照系统的信号流程将各系统模块正确连接起来。连接系统模块的步骤如下：

(1) 将光标指向起始块的输出端口，此时光标变成“+”。

(2) 单击鼠标左键并拖动到目标模块的输入端口，在接近到一定程度时光标变成双十字。这时松开鼠标键，连接完成。完成后在连接点处出现一个箭头，表示系统中信号的流向，如图4.14所示。

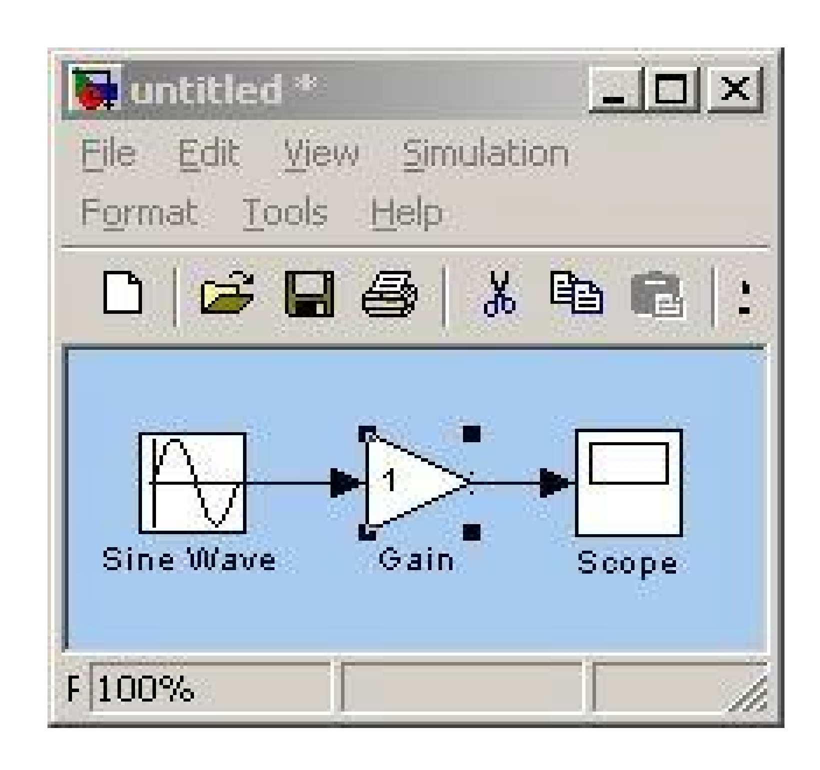


图4.14 系统模块之间的连线

在Simulink的最新版本中，连接系统模块还有如下更有效的方式：

(1) 使用鼠标左键单击起始模块。

(2) 按下Ctrl键，并用鼠标左键单击目标块。

4.3.2模块操作

下面介绍一些对系统模块进行操作的基本技巧，掌握它们可使建立动态系统模型变得更为方便快捷。

1. 模块的复制

如果需要几个同样的模块，可以使用鼠标右键单击并拖动某个块进行拷贝。也可以在选中所需的模块后，使用Edit菜单上的 Copy 和Paste 或使用热键Ctrl+C和Ctrl+V完成同样的功能，如图4.15所示。

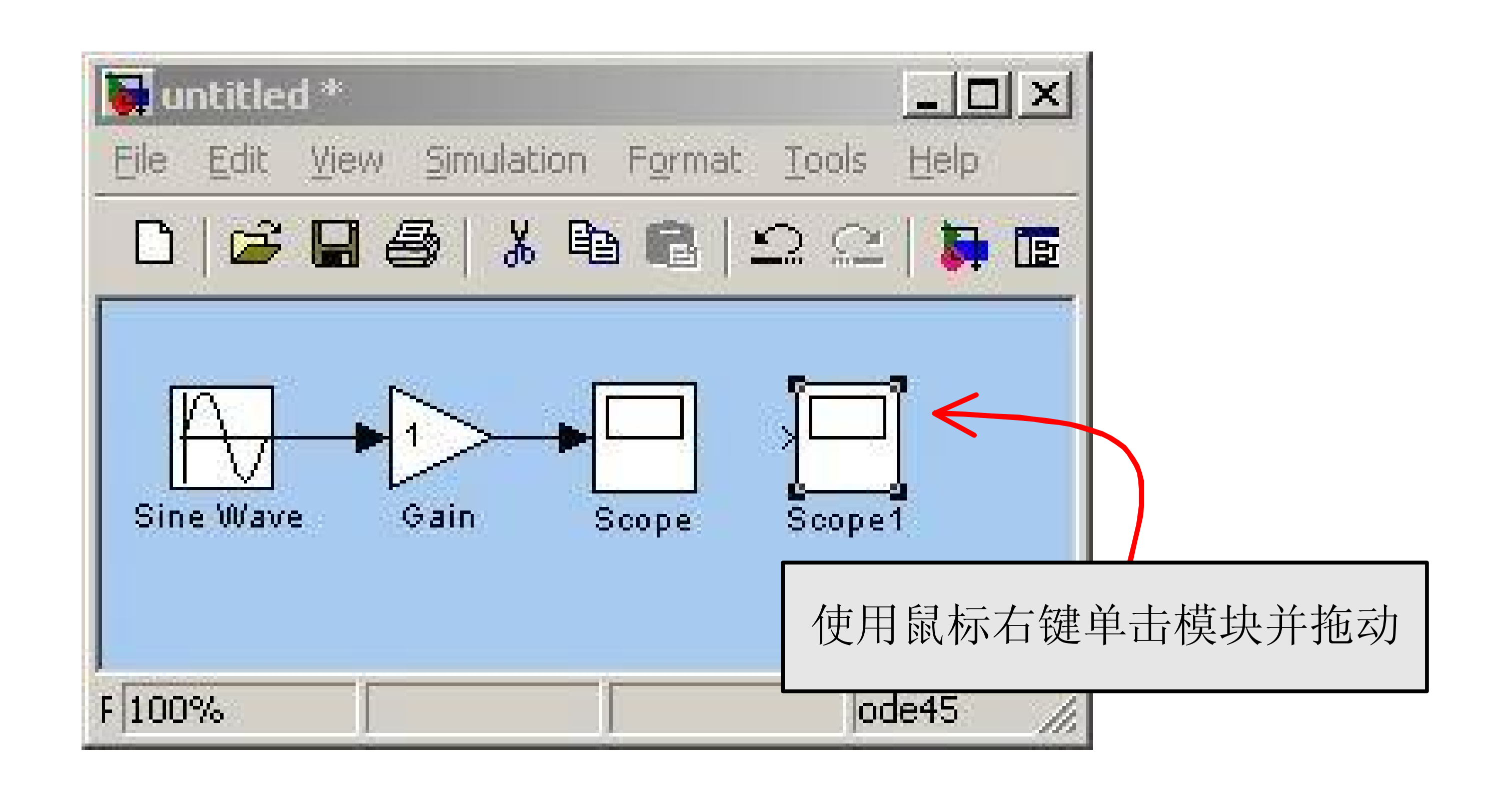

untitled *
File Edit View Simulation Format Tools Help
Sine Wave
1
Gain
Scope
Scope1
使用鼠标右键单击模块并拖动
100%
ode45

图4.15 模块的复制

2. 模块的插入

如果用户需要在信号连线上插入一个模块，只需将这个模块移到线上就可以自动连接。注意这个功能只支持单输入单输出模块。对于其他的模块，只能先删除连线，放置块，然后再重新连线。具体操作如图4.16所示。

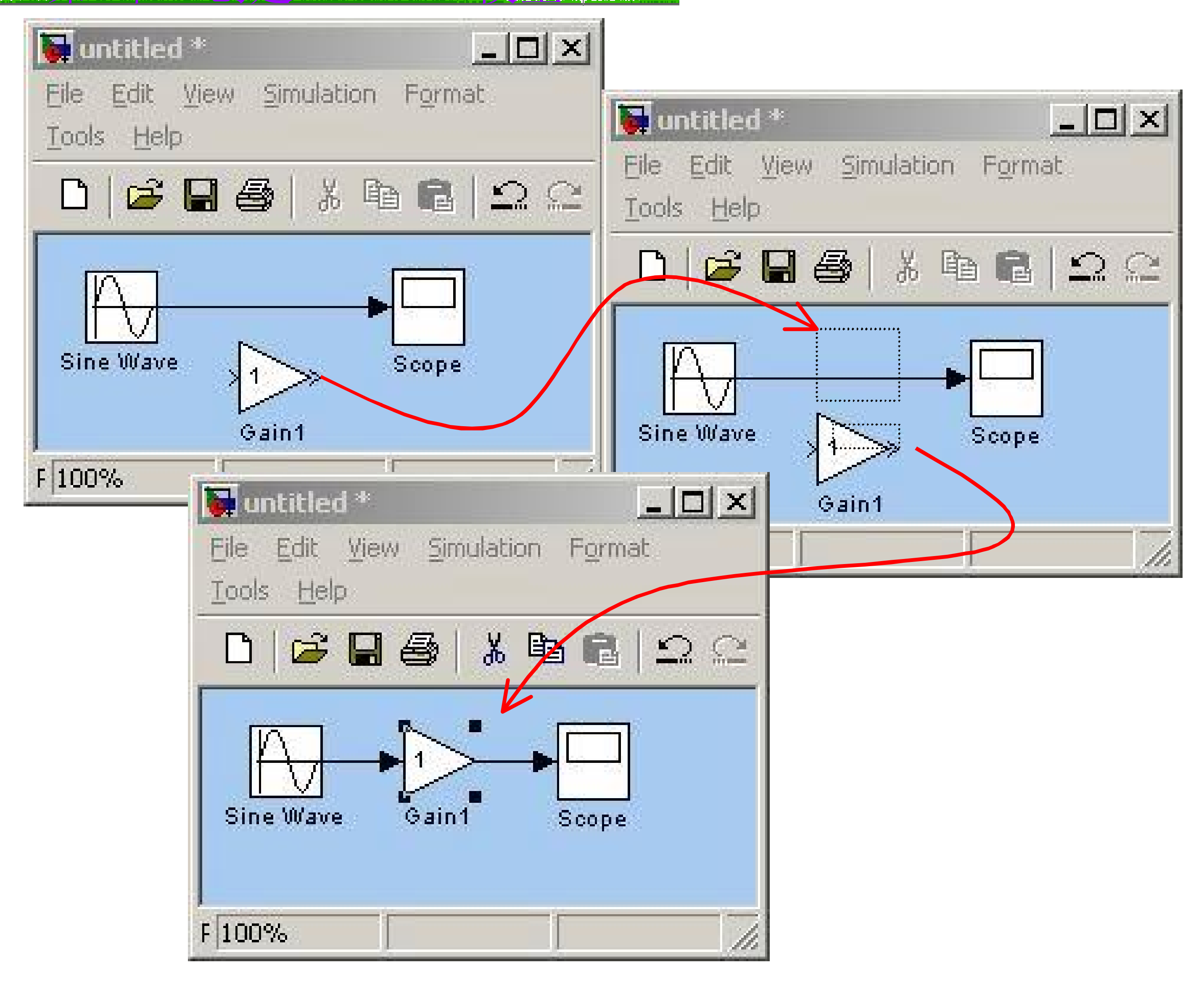


图4.16　系统模块的插入

3. 连线分支与连线改变

在某些情况下，一个系统模块的输出同时作为多个其它模块的输入，这时需要从此模块中引出若干连线，以连接多个其它模块。对信号连线进行分支的操作方式为：使用鼠标右键单击需要分支的信号连线（光标变成“+”），然后拖动到目标模块。

对信号连线还有以下几种常用的操作：

(1) 使用鼠标左键单击并拖动以改变信号连线的路径。

(2) 按下Shift键的同时，在信号连线上单击鼠标左键并拖动，可以生成新的节点。

(3) 在节点上使用鼠标左键单击并拖动，可以改变信号连线路径。

信号连线分支与连线改变如图4.17所示。

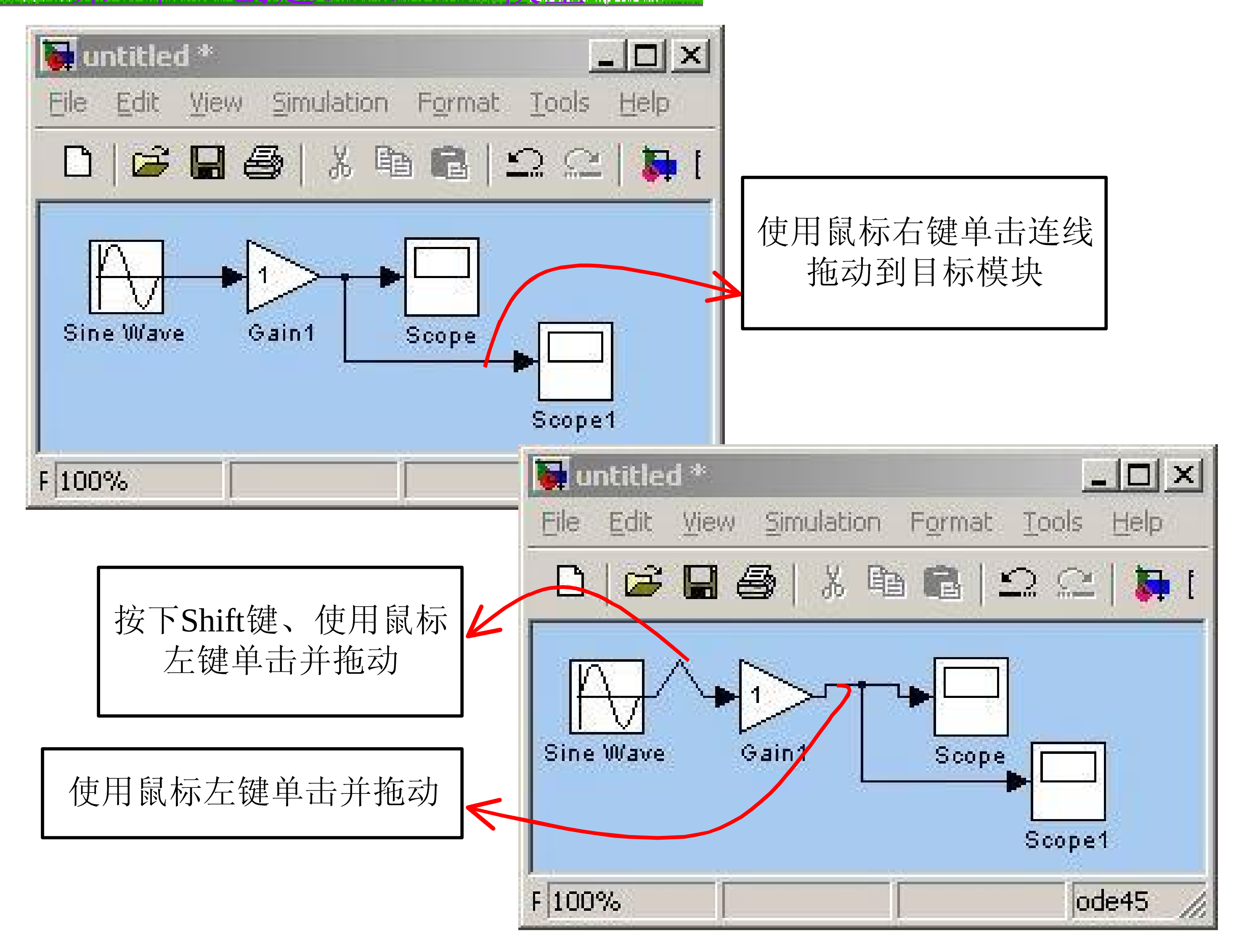


图4.17 连线分支与连线改变

4. 信号组合

在利用Simulink进行系统仿真时，在很多情况下，需要将系统中某些模块的输出信号（一般为标量）组合成一个向量信号，并将得到的信号作为另外一个模块的输入。

图4.18 信号组合

4.3.3 运行仿真

1. 系统模块参数设置与系统仿真参数设置

当用户按照信号的输入输出关系连接各系统模块之后，系统模型的创建工作便已结束。为了对动态系统进行正确的仿真与分析，必须设置正确的系统模块参数与系统仿真参数。系统模块参数的设置方法如下：

(1) 双击系统模块，打开系统模块的参数设置对话框。

(2) 在参数设置对话框中设置合适的模块参数。

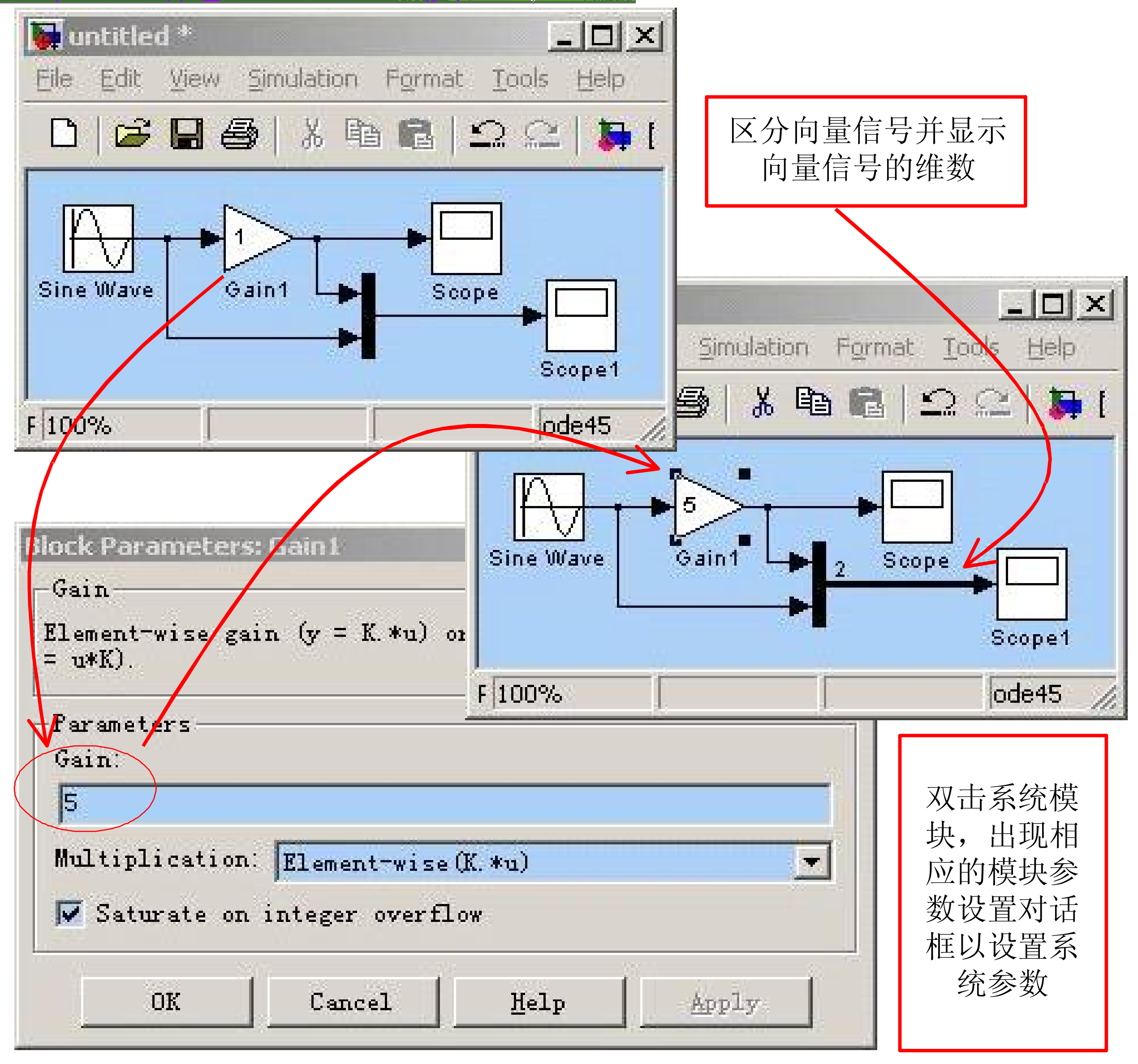


图4.19 系统模块参数设置

当系统中各模块的参数设置完毕后，可设置合适的系统仿真参数以进行动态系统的仿真。有关系统仿真参数设置的知识将在第5章中进行详细的介绍，这里不再赘述。对于图4.19所示的动态系统，系统模块参数设置如图中所示（增益取值为5），系统仿真参数采用Simulink的默认设置。

2. 运行仿真

当对系统中各模块参数以及系统仿真参数进行正确设置之后，单击系统模型编辑器上的Play图标（黑色三角）或选择Simulation菜单下的Start便可以对系统进行仿真分析。对于图4.19所示的动态系统，采用上述的模块参数设置与默认的仿真参数进行仿真。仿真结束后双击Scope模块以显示系统仿真的输出结果，如图4.20所示。

单击 Start Simulation 按钮以缺省参数仿真

向量信号输出，其中黄色显示为 Mux 第一端口的信号、紫色显示为 Mux 第二端口的信号。

图4.20　系统仿真及结果输出



4.4 设计Simulink框图的界面

4.3节中对使用Simulink进行系统建模与仿真做了简单的介绍，任何动态系统的模型构建与仿真的步骤都与此类似。本节所要介绍的Simulink界面设计主要用来改善系统模型的界面，以便于用户对系统模型的理解与维护。

4.4.1 模块及框图属性编辑

1. 框图的视图调整

在Simulink系统模型编辑器中，可以对系统模型的视图进行调整以便更好地观察系统模型。视图调整的方法如下所述：

(1) 使用View菜单控制模型在视图区的显示，用户可以对模型视图进行任意缩放。

(2) 使用系统热键R（放大）或V（缩小）。

(3) 按空格键可以使系统模型充满整个视图窗口。

视图调整效果如图4.21所示。

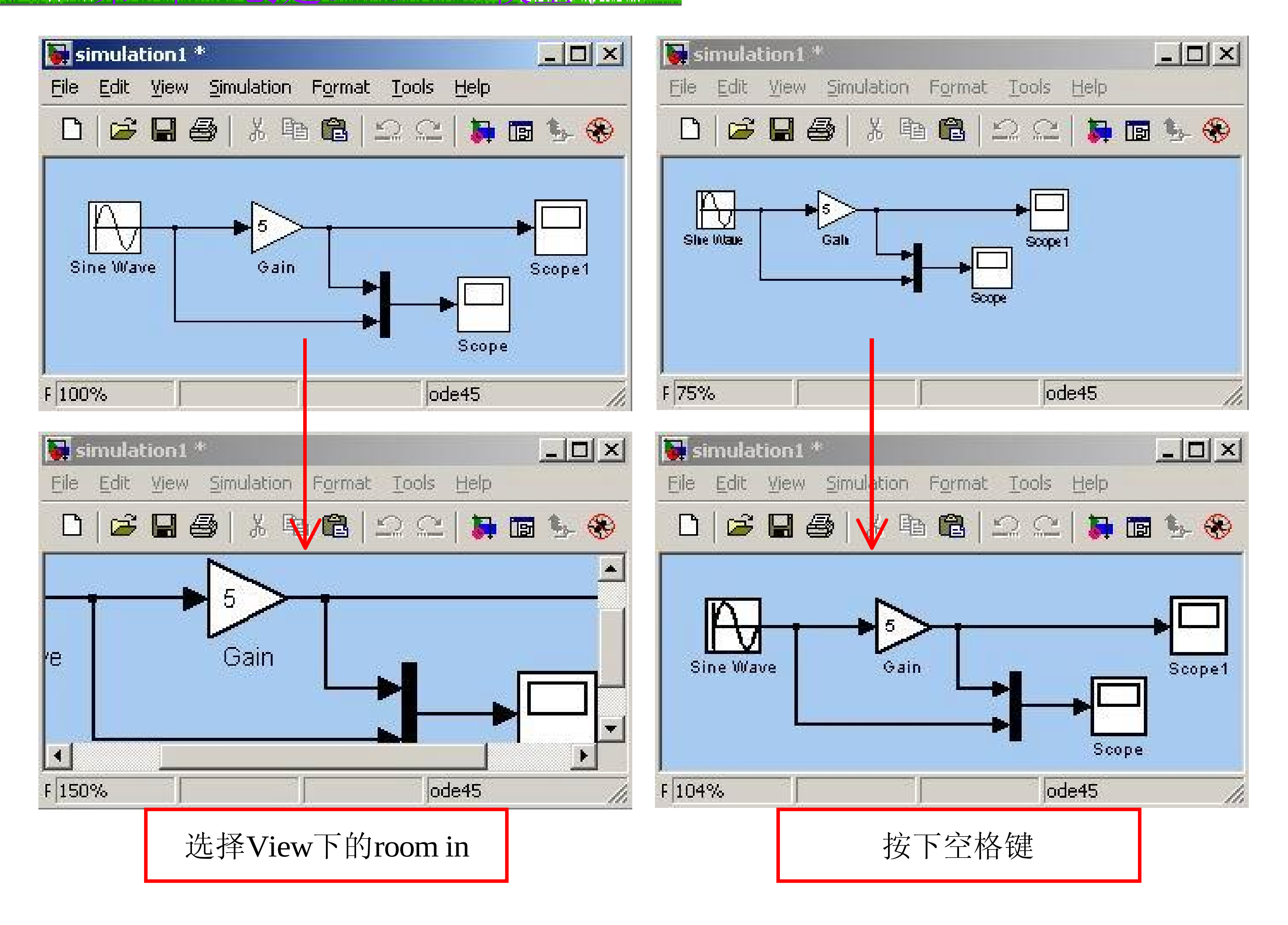


图4.21 改变系统模型的视图

2. 模块的名称操作

在使用Simulink中的系统模块构建系统模型时，Simulink会自动给系统模型中的模块命名，如在4.3节的简单动态系统中，正弦信号模块名称为Sine Wave；对于系统模型中相同的模块，Simulink会自动对其进行编号。一般对于简单的系统，可以采用Simulink的自动命名；但对于复杂系统，给每个模块取一个具有明显意义的名称非常有利于系统模型的理解与维护。下面简单介绍一下模块名称的操作。

(1) 模块命名：使用鼠标左键单击模块名称，进入编辑状态，然后键入新的名称。

(2) 名称移动：使用鼠标左键单击模块名称并拖动到模块的另一侧，或选择Format菜单中的Flip Name翻转模块名称。

(3) 名称隐藏：选择Format菜单中的Hide Name隐藏系统模块名称。

注意，系统模型中模块的名称应当是唯一的，否则Simulink会给出警告并自动改变名称。系统模型中模块的名称操作如图4.22所示。

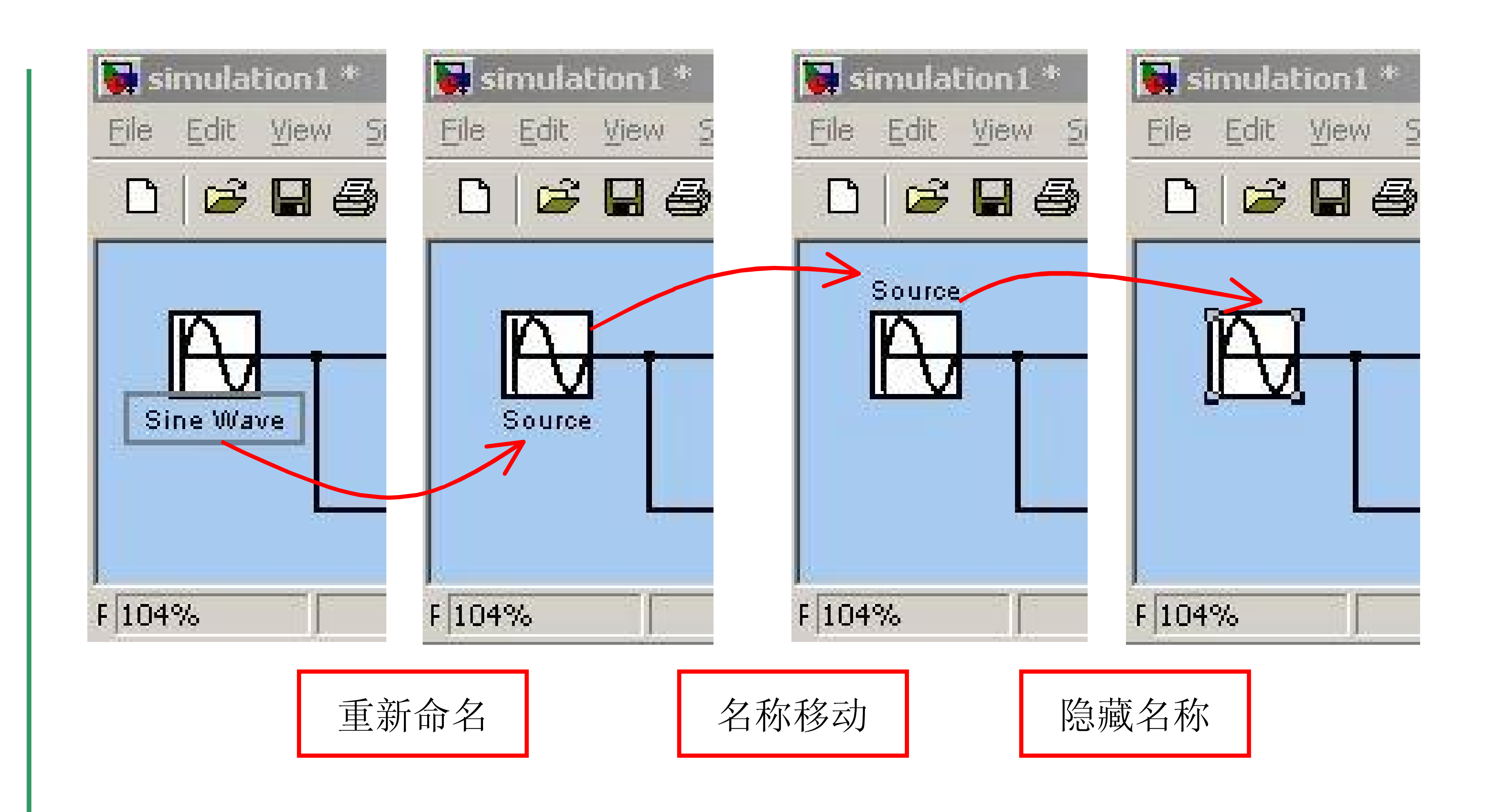


图4.22 系统模型中模块的名称操作

3. 模块的其它操作

Simulink允许用户对模块的几何尺寸进行修改，以改善系统模型框图的界面。例如，对于具有多个输入端口的模块，需要调整其大小使其能够较好地容纳多个信号连线，而非采用模块的默认大小；另外，对于某些系统模块，当模块的尺寸足够大时，模块的参数将直接显示在模块上面，这非常有利于用户对模型的理解。

Simulink允许改变模块的颜色。使用鼠标右键单击模块，选择Foreground color或Background color菜单来设置颜色；或使用模型编辑器中Format菜单中的相应命令设置模块颜色。如果模块的前景色发生改变，则所有由此模块引出的信号线颜色也随之改变；当系统模型框图很复杂时，这个特性能够有效地增强框图的可读性。

此外，还可以使用Format菜单中的Show Drop Shadow为模块生成阴影，或使用Flip Block、Rotate Block对模块进行翻转与旋转，或使用Font对模块字体进行设置等。对模块的操作如图4.23所示。

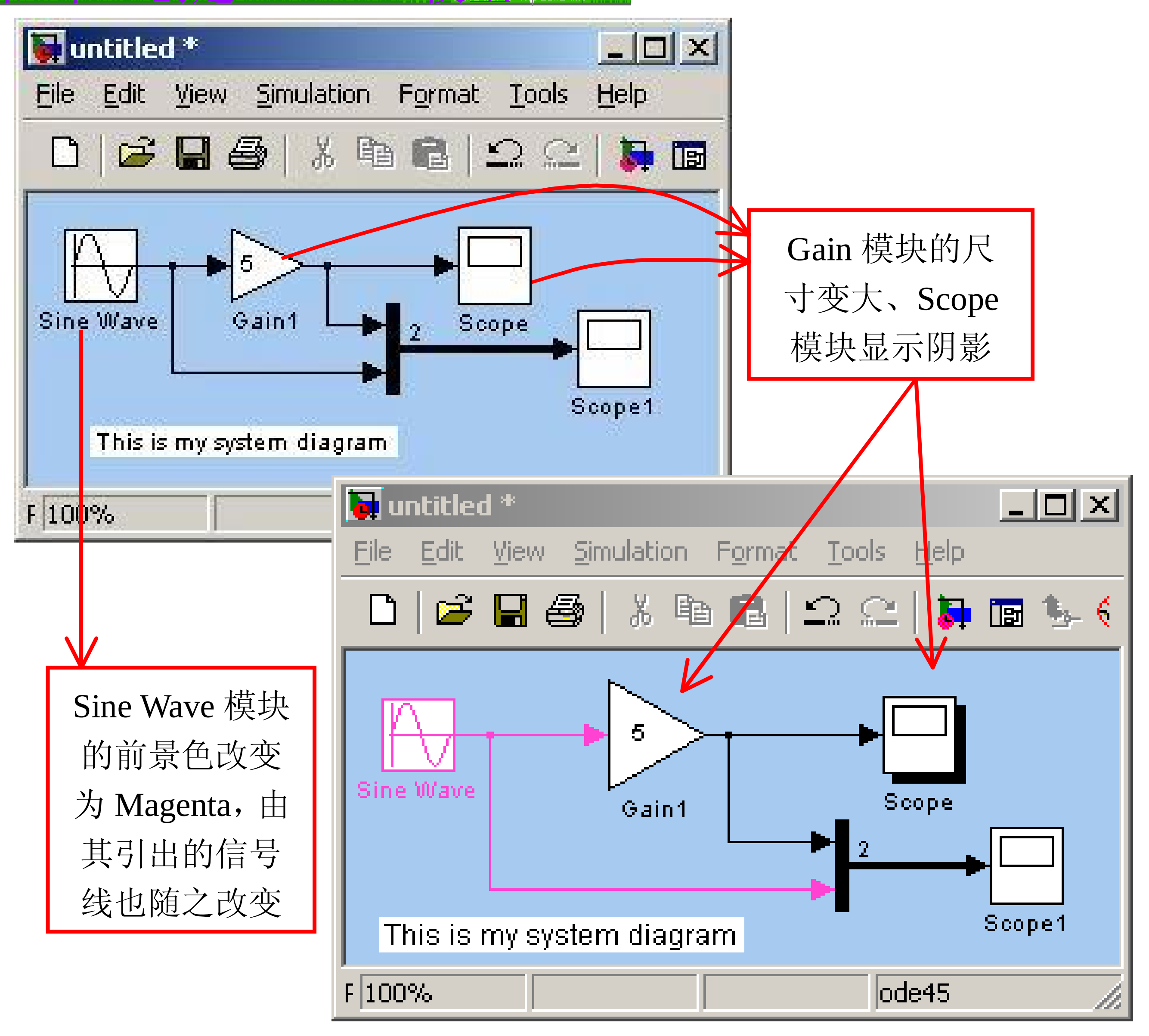


图4.23 模块的其它操作

4. 系统框图注释

作为友好的Simulink系统模型界面，对系统模型的注释是不可缺少的。在Simulink中对系统模型框图进行注释的方法非常简单，只需在系统模型编辑器的背景上双击鼠标左键以确定添加注释文本的位置，并打开一个文本编辑框，用户便可以在此输入相应的注释文本。输入完毕后，使用鼠标左键单击以退出编辑并移动文本位置（编辑框未被选中情况下）到合适的地方。此外，在文本对象上单击鼠标右键，可以改变文本的属性如大小、字体和对齐方式等。在任何时候都可以双击注释文本进行编辑。系统框图注释如图4.24所示。

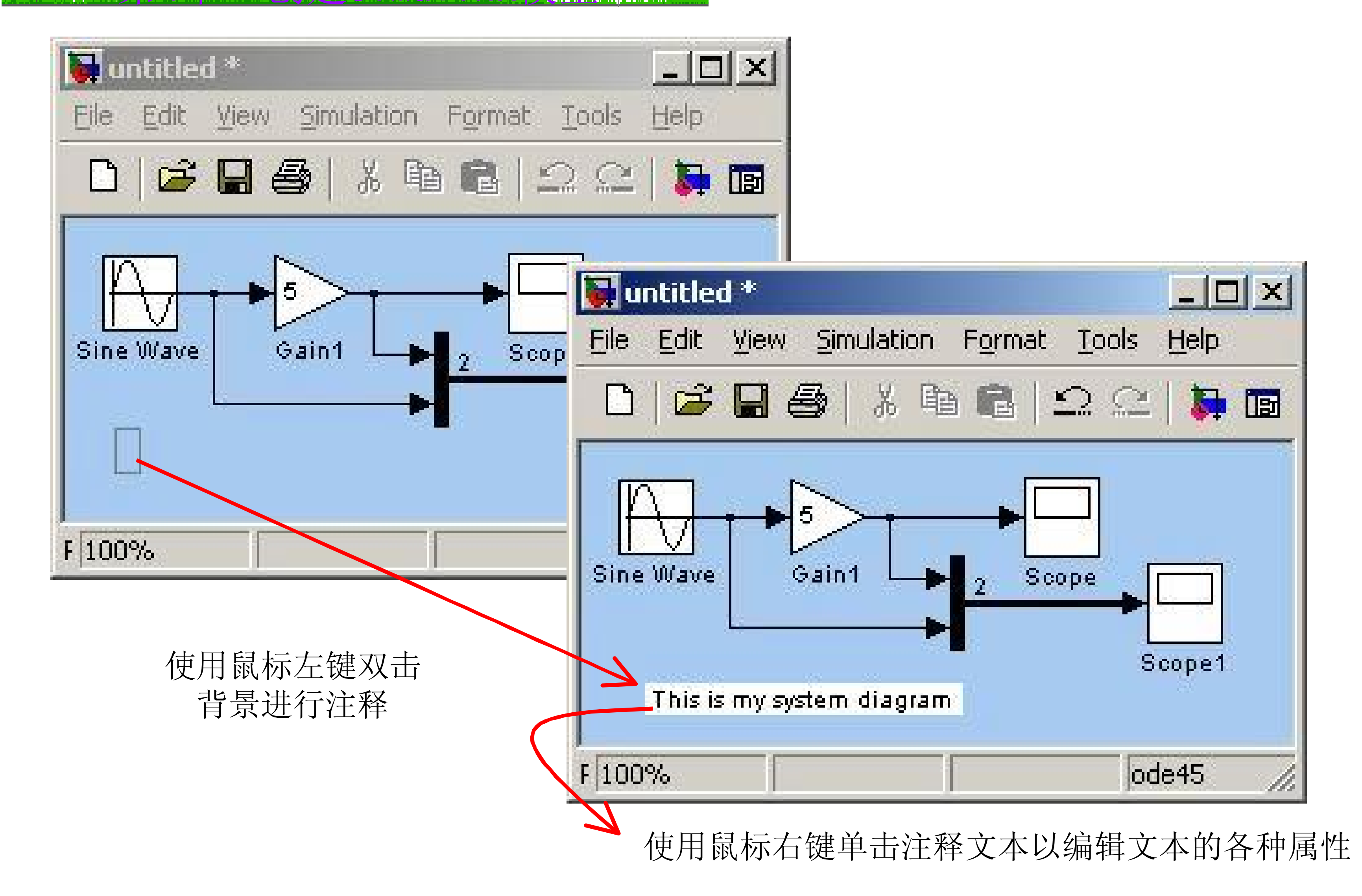


图4.24 系统模型框图注释

4.4.2信号标签与标签传递

1. 信号标签

在创建系统模型尤其是大型复杂系统模型时，信号标签对理解系统框图尤为重要。所谓的信号标签，也可以称为信号的“名称”或“标记”，它与特定的信号相联系，是信号的一个固有属性。这一点与系统框图注释不同，框图注释是对整个或局部系统模型进行说明的文字信息，它与系统模型相分离。

生成信号标签的方法有如下两种：

(1) 使用鼠标左键双击需要加入标签的信号（即系统模型中与信号相对应的模块连线），这时便会出现标签编辑框，在其中键入标签文本即可。与框图注释类似，信号标签可以移动到希望的位置，但只能是在信号线的附近。如果强行将标签拖动离开信号线，标签会自动回到原处。当一个信号定义了标签后，从这条信号线引出的分支线会继承这个标签，如图4.25所示。

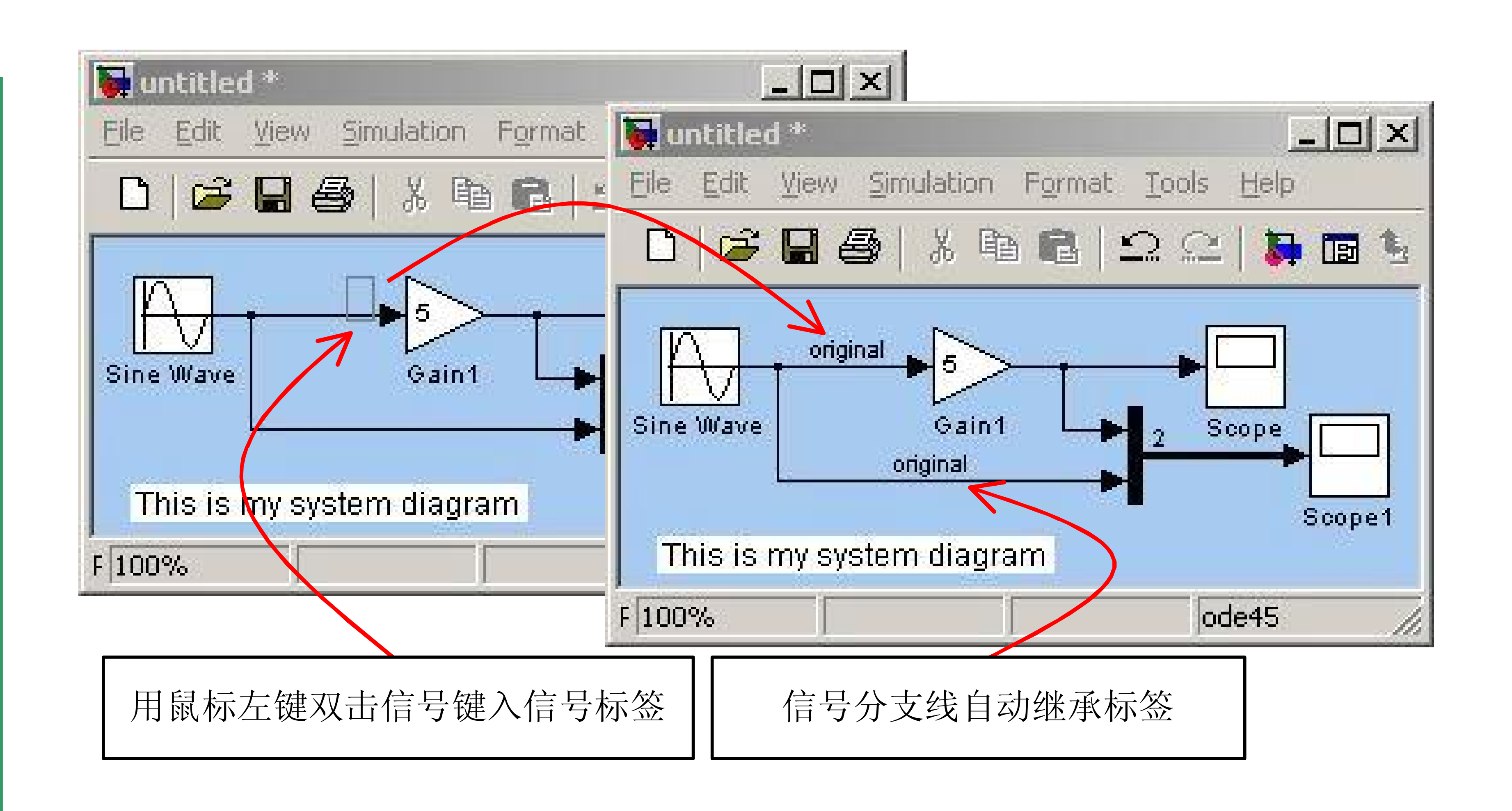


图4.25 信号标签操作之一

(2) 首先选择需要加入标签的信号，用鼠标左键单击信号连线；然后使用Edit菜单下的 Signal Properties项，在打开的界面中编辑信号的名称，而且还可以使用这个界面对信号作简单的描述并建立HTML文档链接，如图4.26所示。

注意，虽然信号标签的内容可以任意指定，但为了系统模型可读性，信号标签最好使用能够代表信号特征的名称（如信号类型、信号作用等）。

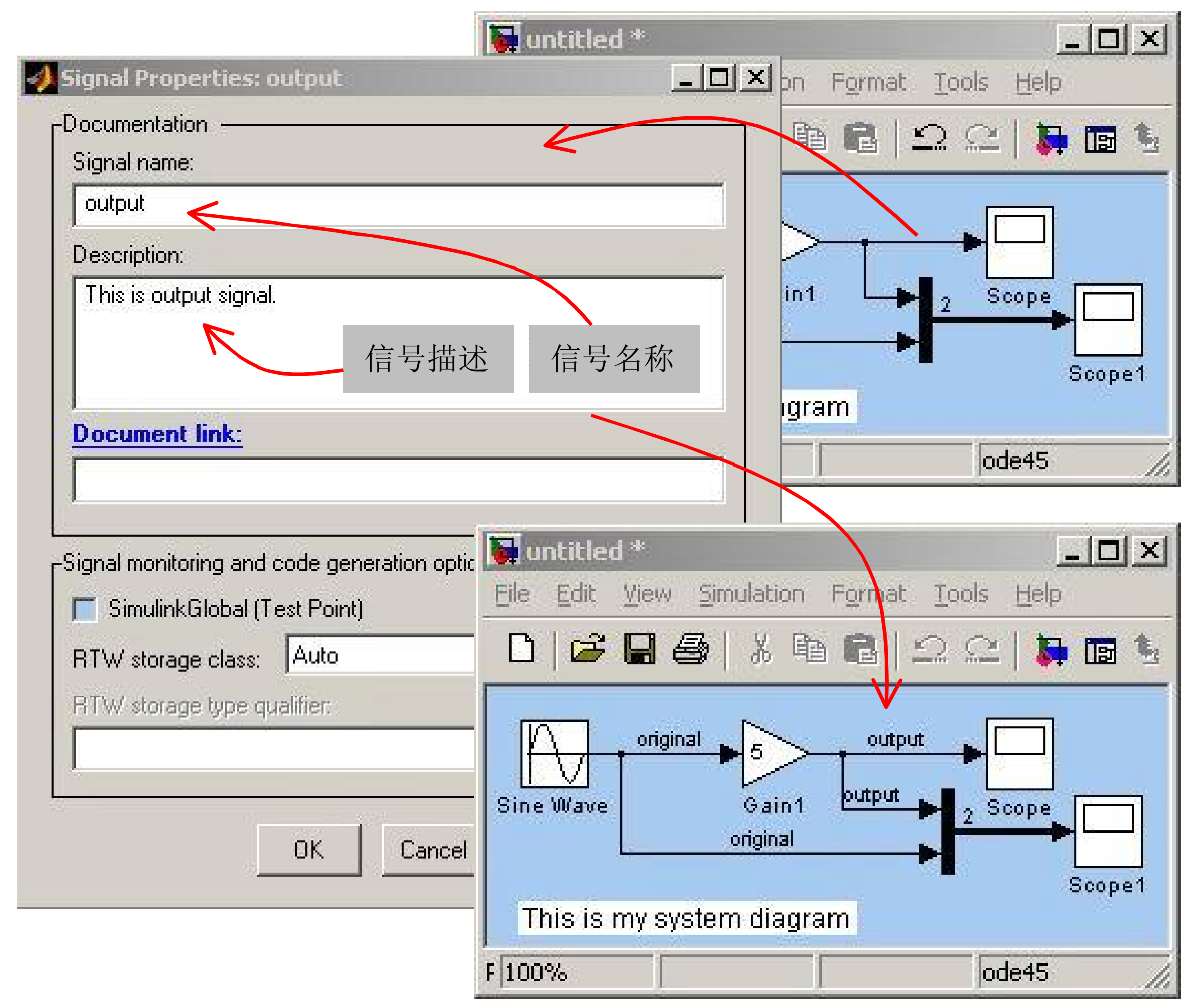


图4.26 信号标签操作之二

2. 信号标签的传递

在系统模型中，信号标签可以由某些称之为“虚块”的系统模块来进行传递。这些虚块主要用来完成对信号的选择、组合与传递，它不改变信号的任何属性。如Signals & Systems模块库中的Mux模块的功能是组合信号，但并不改变信号的值。

信号标签传递的方法有如下几种：

(1) 选择信号线并用鼠标左键双击，在信号标签编辑框中键入< >,在此尖括号中键入信号标签即可传递信号标签。

(2) 选择信号线，然后选择Edit菜单中的Signal Properties；或单击鼠标右键，选择弹出式菜单中的Signal Properties，将Show Propagated Signals设置为 on即可。

注意：只能在信号的前进方向上传递该信号标签。当一个带有标签的信号与Scope块连接时，信号标签将作为标题显示。信号标签的传递如图4.27所示。

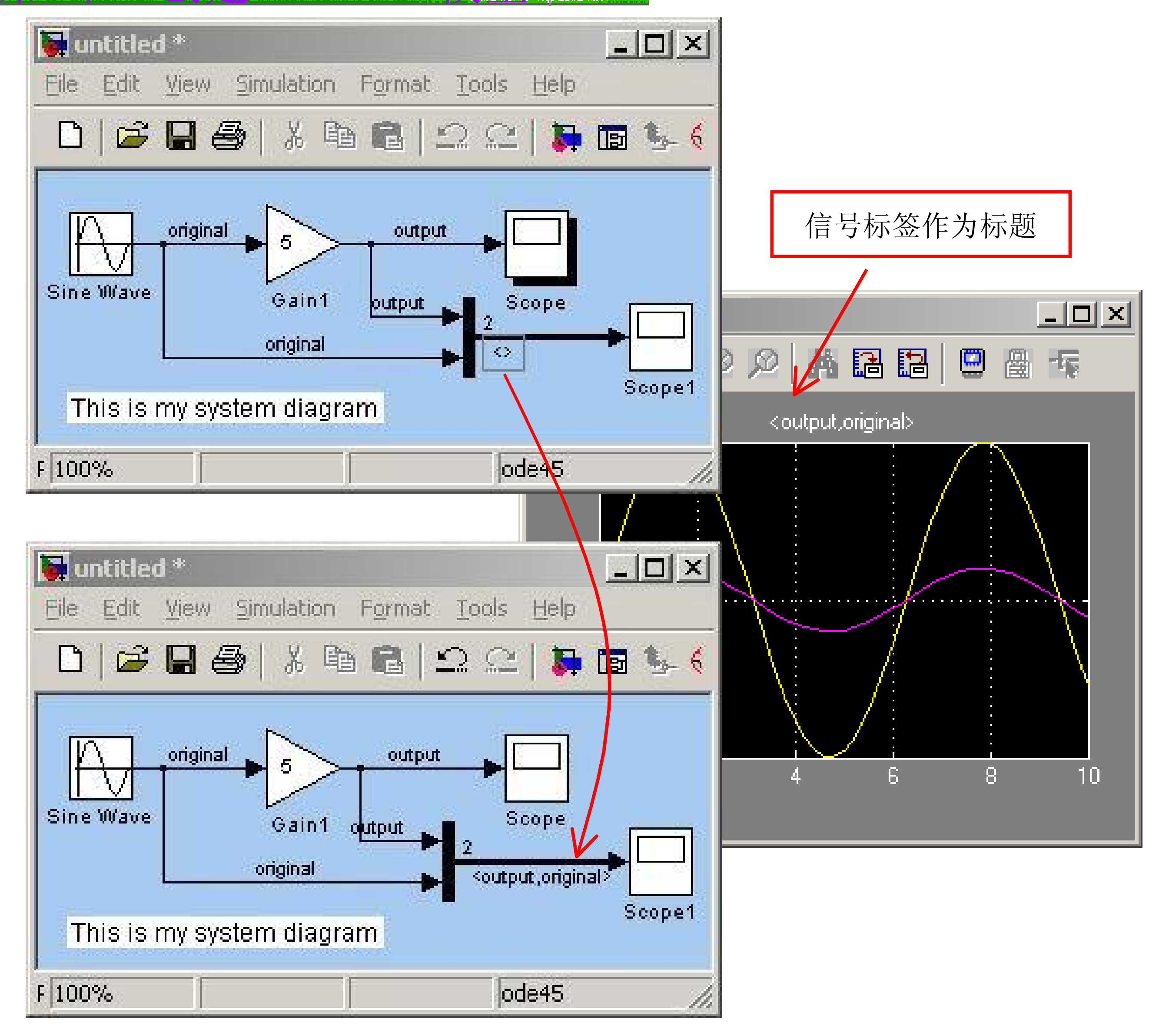


图4.27 信号标签的传递

4.4.3 Simulink子系统介绍

对于简单的动态系统而言，用户很容易建立系统模型并分析系统模型中各模块之间的相互关系，以及模块的输入输出关系。但是对于比较复杂的系统，系统模型中包含的模块数目较多，模块之间的输入输出关系比较复杂。这时对于分析与设计系统而言，都会给用户带来诸多的不便，而使用子系统技术则可以较好地解决这一问题

1. 子系统生成

Simulink提供的子系统功能可以大大地增强Simulink系统模型框图的可读性。所谓的子系统可以理解为一种“容器”，此容器能够将一组相关的模块封装到一个单独的模块中，并且与原来系统模块组的功能一致。

子系统的建立方法有如下两种：

(1) 在已有的系统模型中建立子系统：首先框选待封装的区域，即在模型编辑器背景中单击鼠标左键并拖动，选中需要放置到子系统中的模块与信号（或在按下Shift键的同时，用鼠标左键单击所需模块）；然后选择Edit菜单下的Create Subsystem，即可建立子系统。如图4.28所示。

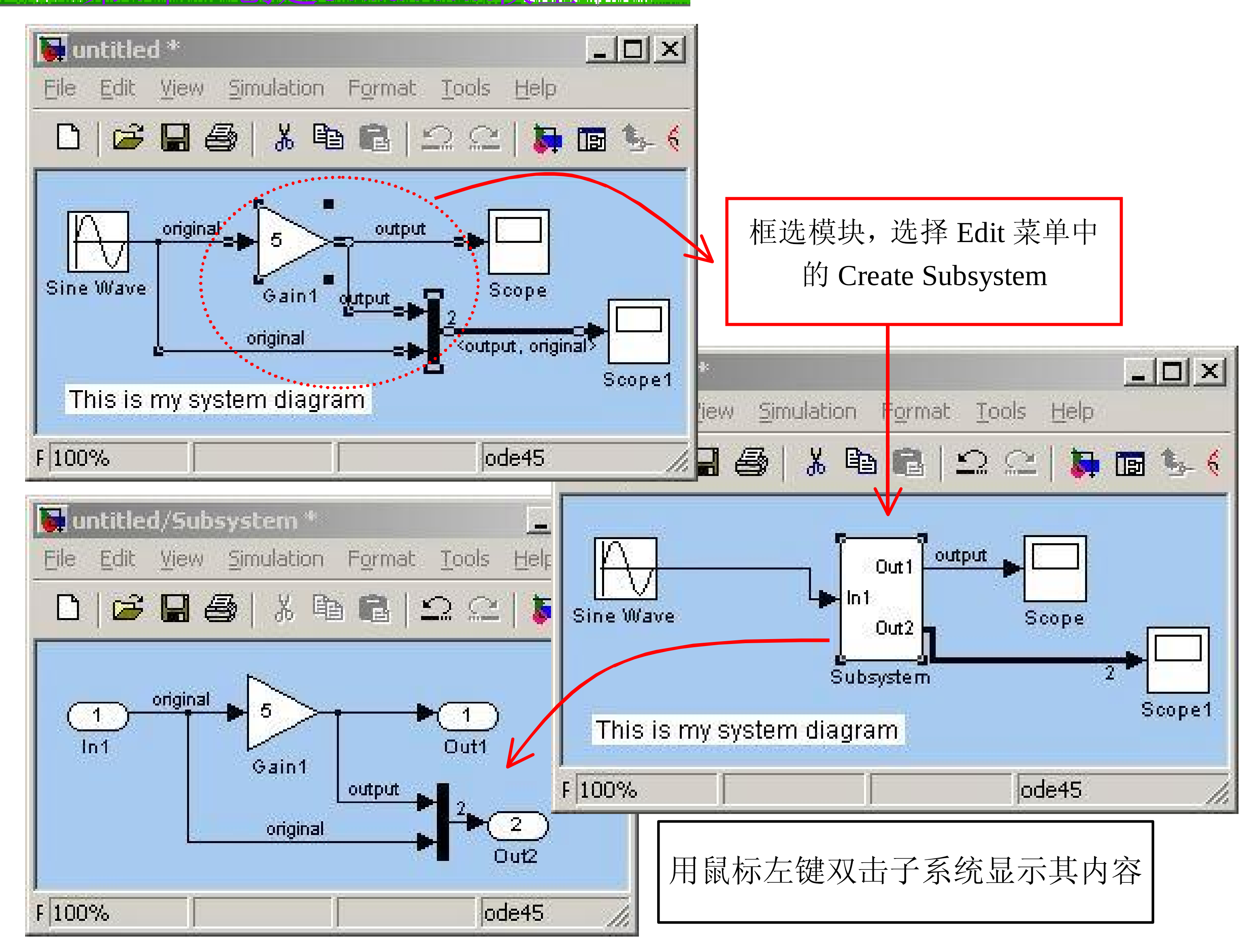


图4.28 子系统建立：选择模块生成子系统

(2) 建立空的子系统：使用Subsystems模块库中的模块建立子系统。这样建立的子系统内容为空，然后双击子系统对其进行编辑。如图4.29所示。

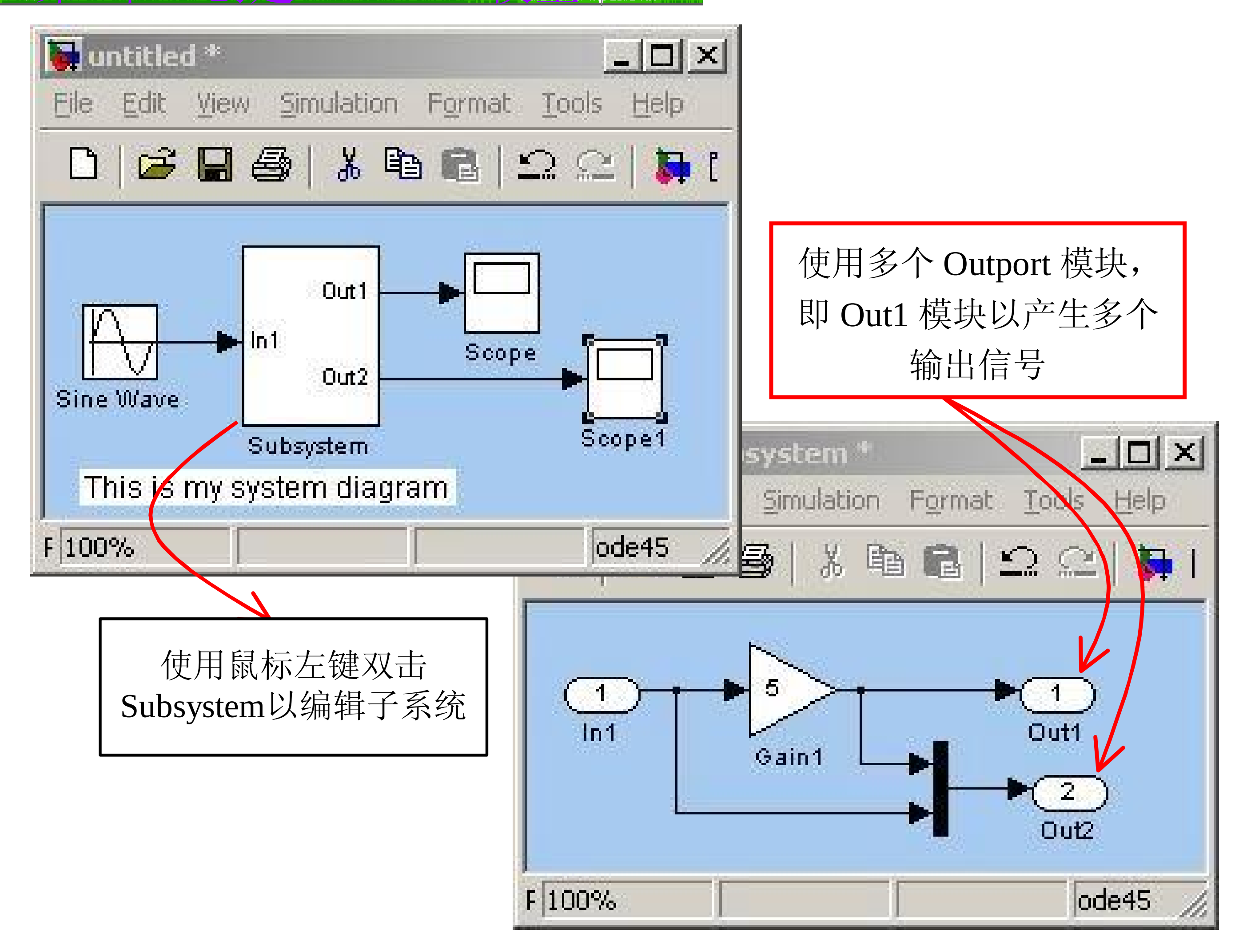


图4.29 子系统建立：生成并编辑空子系统

建立此系统模型所需要的系统模块如下所述：Subsystems模块库中的Subsystem模块、Sources模块库中的Sine Wave、Sinks模块库中的Scope模块、Sinks模块库中的Out1模块（Subsystem模块的缺省设置为单输入单输出，使用Out1模块可以产生多个输出）、Math模块库中的Gain模块以及Signals & Systems模块库中的Mux模块等。

2. 子系统操作

在生成子系统之后，用户可以对子系统进行各种与系统模块相类似的操作，这时子系统相当于具有一定功能的系统模块。例如，子系统的命名、子系统视图的修改、子系统的显示颜色等等。当然子系统也有其特有的操作，如子系统的显示（用鼠标左键双击子系统模块即可打开子系统）、子系统的封装（将在第7章中进行详细介绍）等等。

3. Inport输入模块与Outport输出模块

在系统模型中建立子系统时，Simulink会自动生成Inport模块（Sources模块库中的In1模块）与Outport模块（Sinks模块库中的Out1模块）。Inport模块作为子系统的输入端口，Outport作为子系统的输出端口，它们被用来完成子系统和主系统之间的通讯。

Inport和Outport用来对信号进行传递，不改变信号的任何属性；另外，信号标签可以越过它们进行传递。如果需要建立多输入多输出的子系统，则需要使用多个Inport模块与Outport模块，而且最好使用合适的名称对Inport模块与Outport模块进行命名，如图4.30所示。

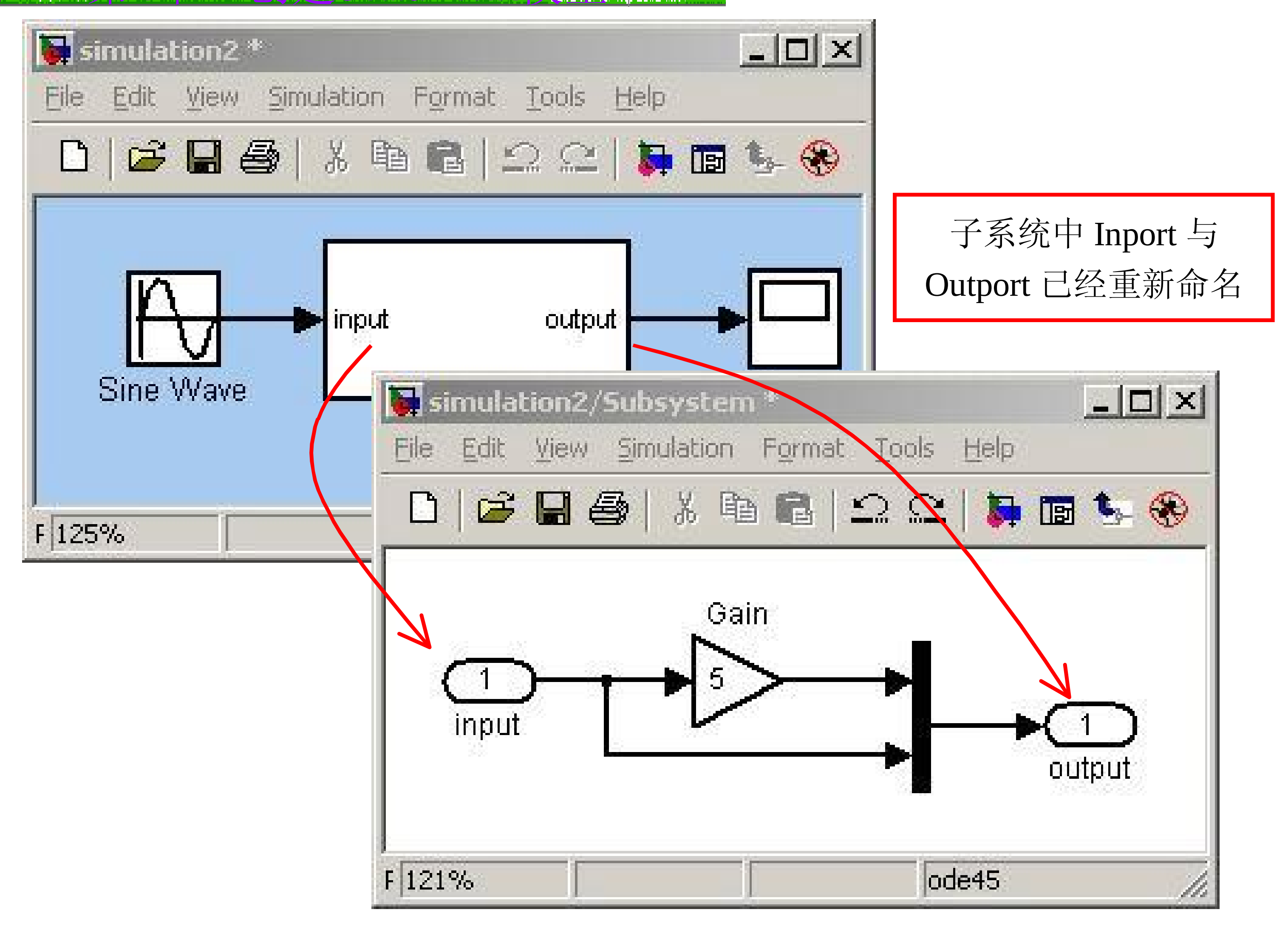


图4.30 Inport模块与Outport模块的重新命名

4.4.4建立复杂系统模型

Simulink适合建立大型复杂系统的模型，它为仿真系统模型的界面组织与设计提供了强大的支持。一般而言，建立复杂系统模型有两种不同的思路：

(1) 自下向上的设计思路：如果用户从草图开始建立一个复杂的模型，可以先建底层模型，然后对已经建好的块生成子系统。

(2) 自顶向下的设计思路：首先设计系统的总体模型，然后再进行细节设计。采用这种方法，可以在顶层使用空的子系统块，然后再实现具体的细节。

图4.31　模型浏览器的使用



4.5 Simulink与MATLAB的接口设计

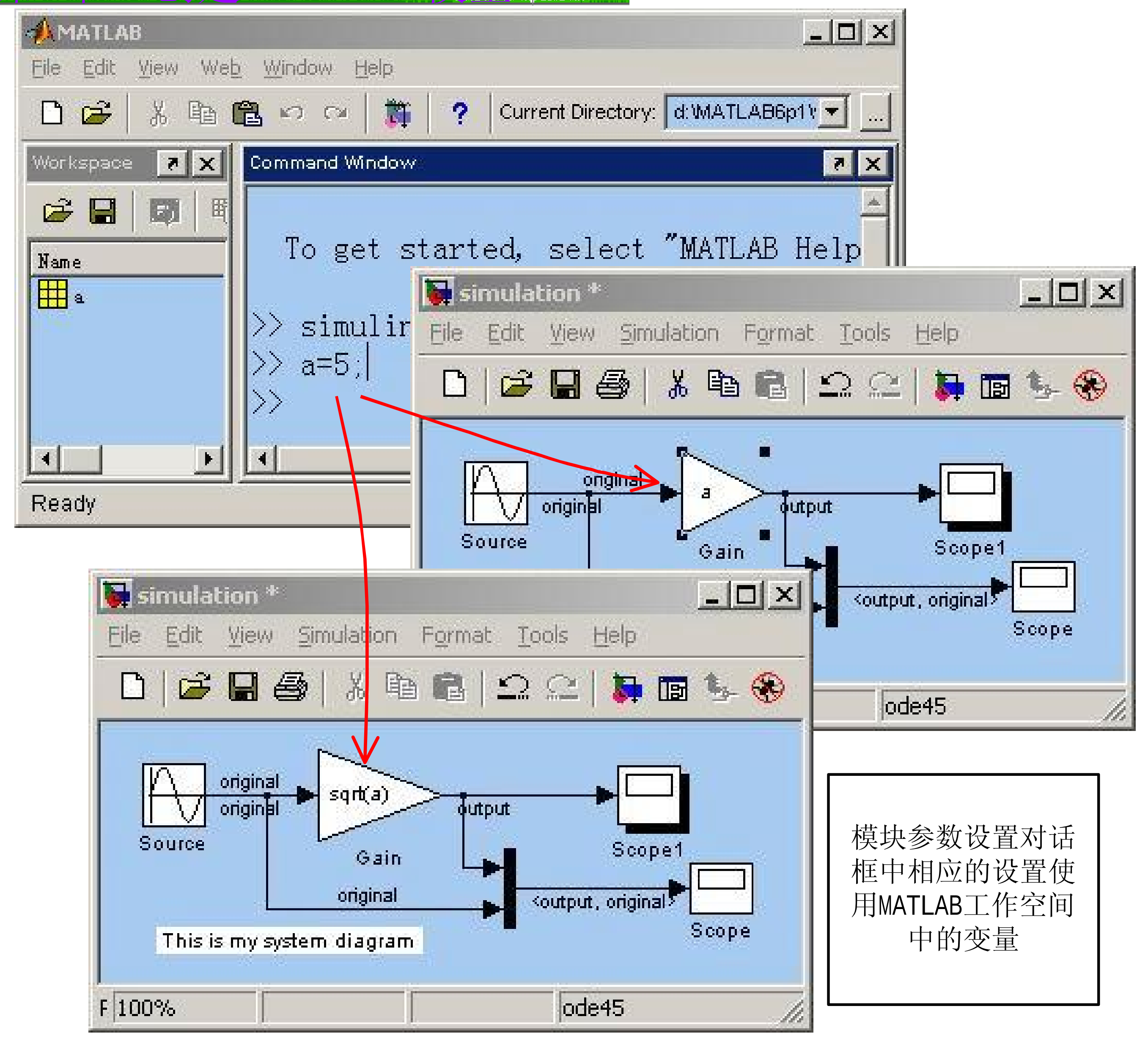
4.5.1 由MATLAB工作空间变量设置系统模块参数

如前所述，用户可以双击一个模块以打开模块参数设置对话框，然后直接输入数据以设置模块参数。其实，用户也可以使用MATLAB工作空间中的变量设置系统模块参数，这对于多个模块的参数均依赖于同一个变量时非常有用。由MATLAB工作空间中的变量设置模块参数的形式有如下两种：

(1) 直接使用MATLAB工作空间中的变量设置模块参数。

(2) 使用变量的表达式设置模块参数。

例如，如果a是定义在MATLAB中的变量，则表达式a、a^2+5、exp(–a)等均可以作为系统模块的参数，如图4.32所示。



模块参数设置对话框中相应的设置使用MATLAB工作空间中的变量

图4.32 使用MATLAB工作空间变量设置模块参数

4.5.2 将信号输出到MATLAB工作空间中

使用示波器模块Scope的输出信号，可以使用户对输出的信号进行简单的定性分析。

使用Sinks模块库中的To Workspace 模块，可以轻易地将信号输出到MATLAB工作空间中。信号输出的名称在To Workspace模块的对话框中设置，此对话框还可以设置输出数据的点数、输出的间隔，以及输出数据的类型等。其中输出类型有三种形式：数组、结构以及带有时间变量的结构。仿真结束或暂停时信号被输出到工作空间中，如图4.33所示。

使用Sinks模块库中的To Workspace模块输出系统模型中指定的信号至MATLAB工作空间

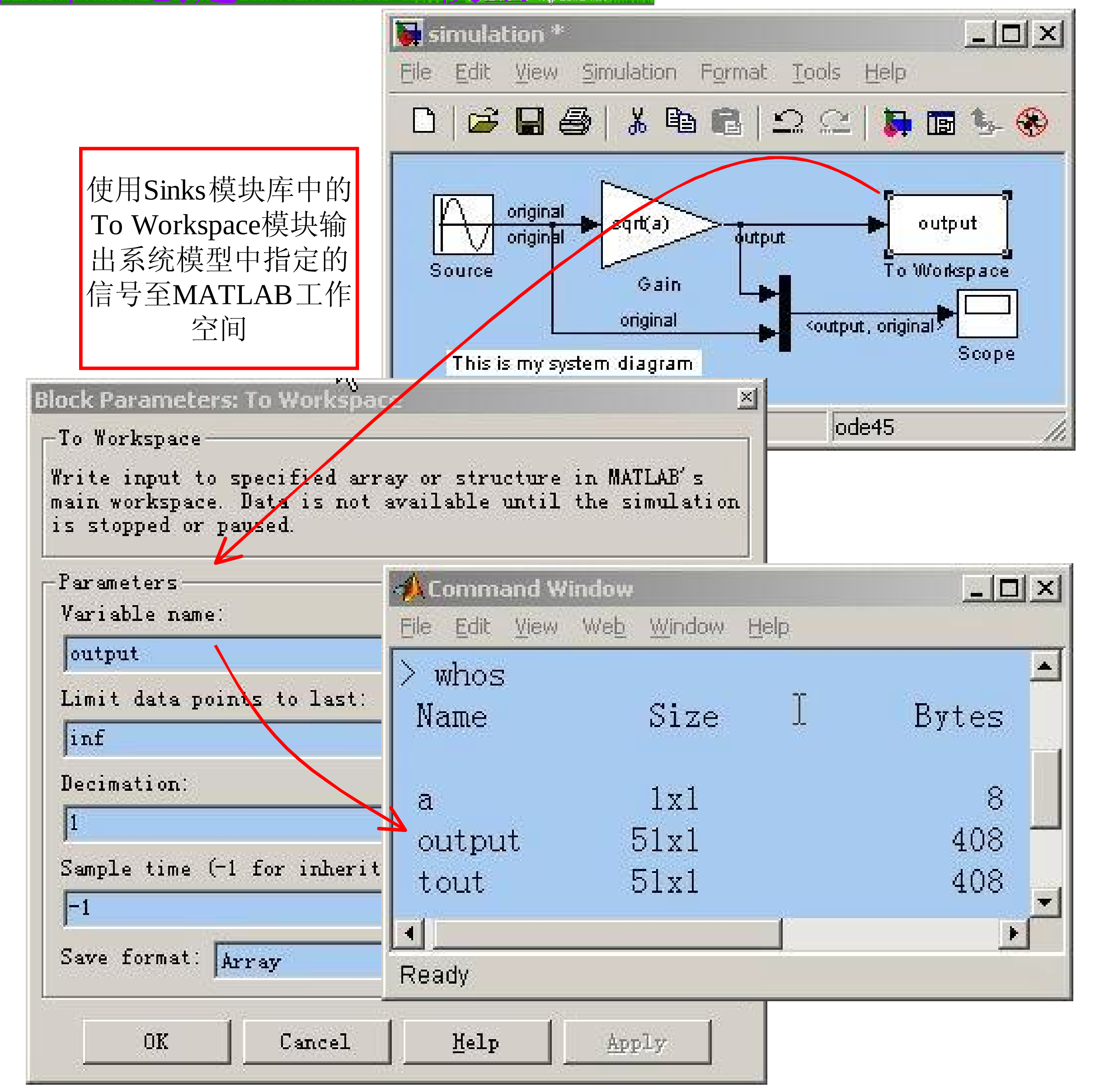


图4.33 系统模型中信号输出

4.5.3 使用工作空间变量作为系统输入信号

Simulink与MATLAB的数据交互是相互的，除了可以将信号输出到MATLAB工作空间中之外，用户还可以使用MATLAB工作空间中的变量作为系统模型的输入信号。使用Sources模块库中的From Workspace模块可以将MATLAB工作空间中的变量作为系统模型的输入信号。此变量的格式如下所示：

>>t=0:time_step:final_time; % 表示信号输入时间范围与时间步长

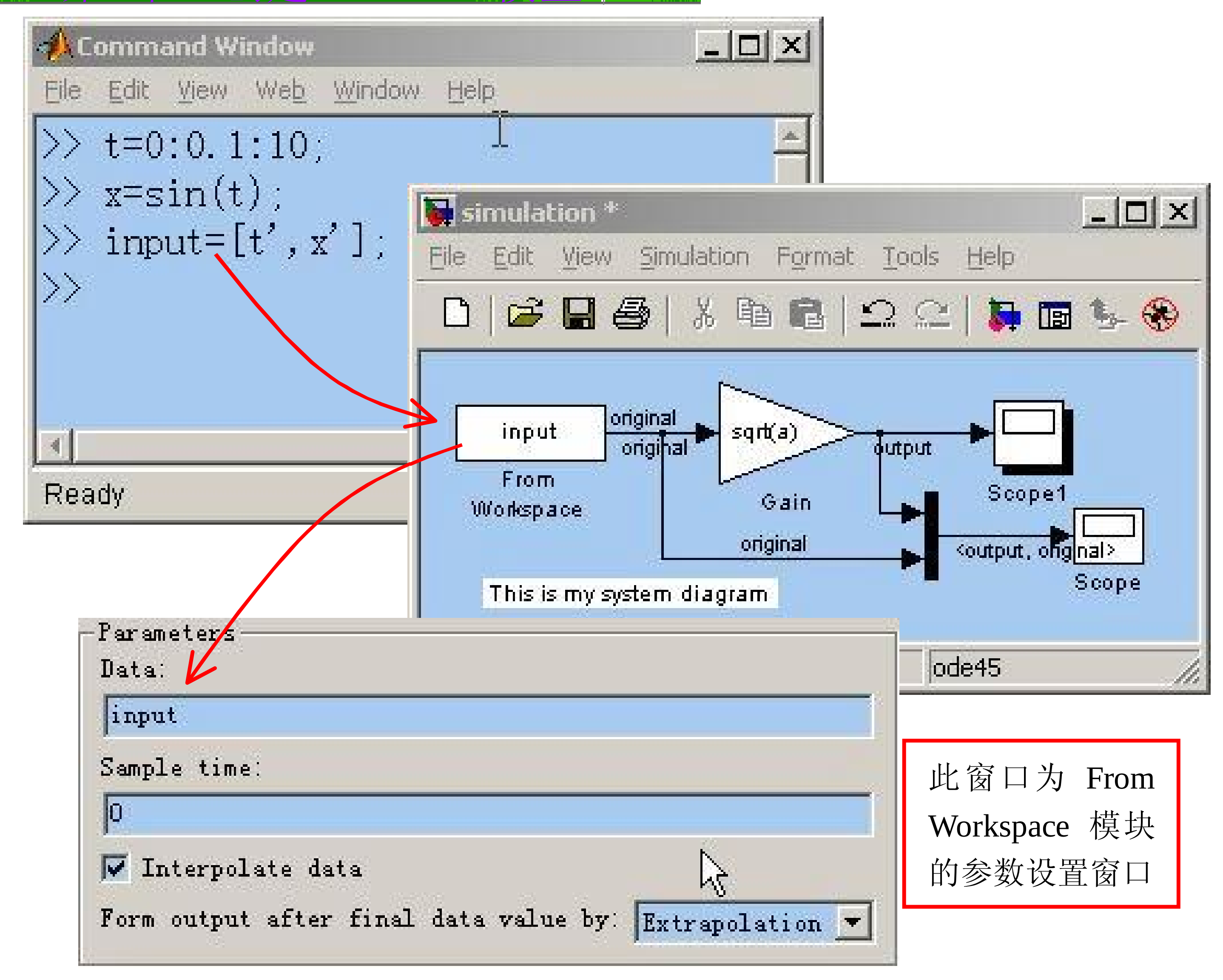
>>x=func(t);   % 表示在每一时刻的信号值

>>input=[t',x'];

%表示信号的输入向量，输入变量第一列须为时间序列，接下来的各列代表信号的取值

例如，在MATLAB命令窗口中键入如下的语句并运行。

```
>>t=0:0.1:10;
>>x=sin(t);
>>input=[t',x'];
```



此窗口为 From Workspace 模块的参数设置窗口

图4.34 MATLAB工作空间变量作为系统输入信号

运行此系统进行仿真，系统输入信号input的作用相当于Sources模块中的Sine Wave模块，其结果如图4.35所示（Scope1显示结果）。

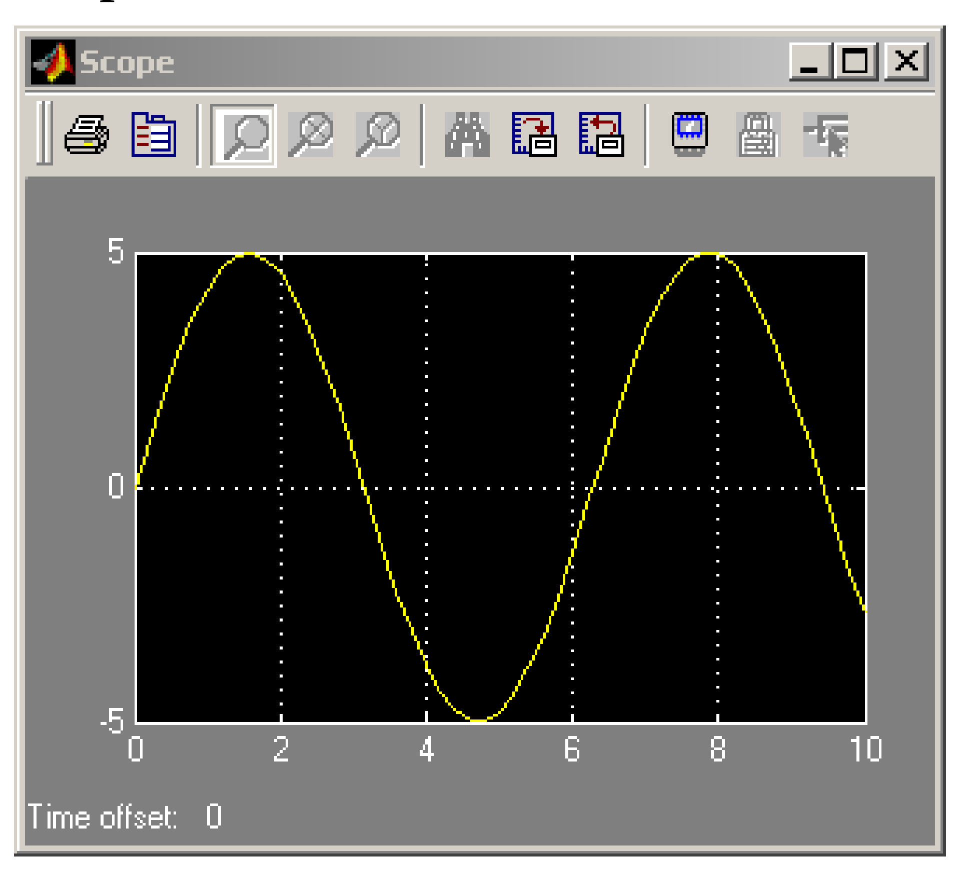


图4.35 使用input信号作为输入的仿真结果

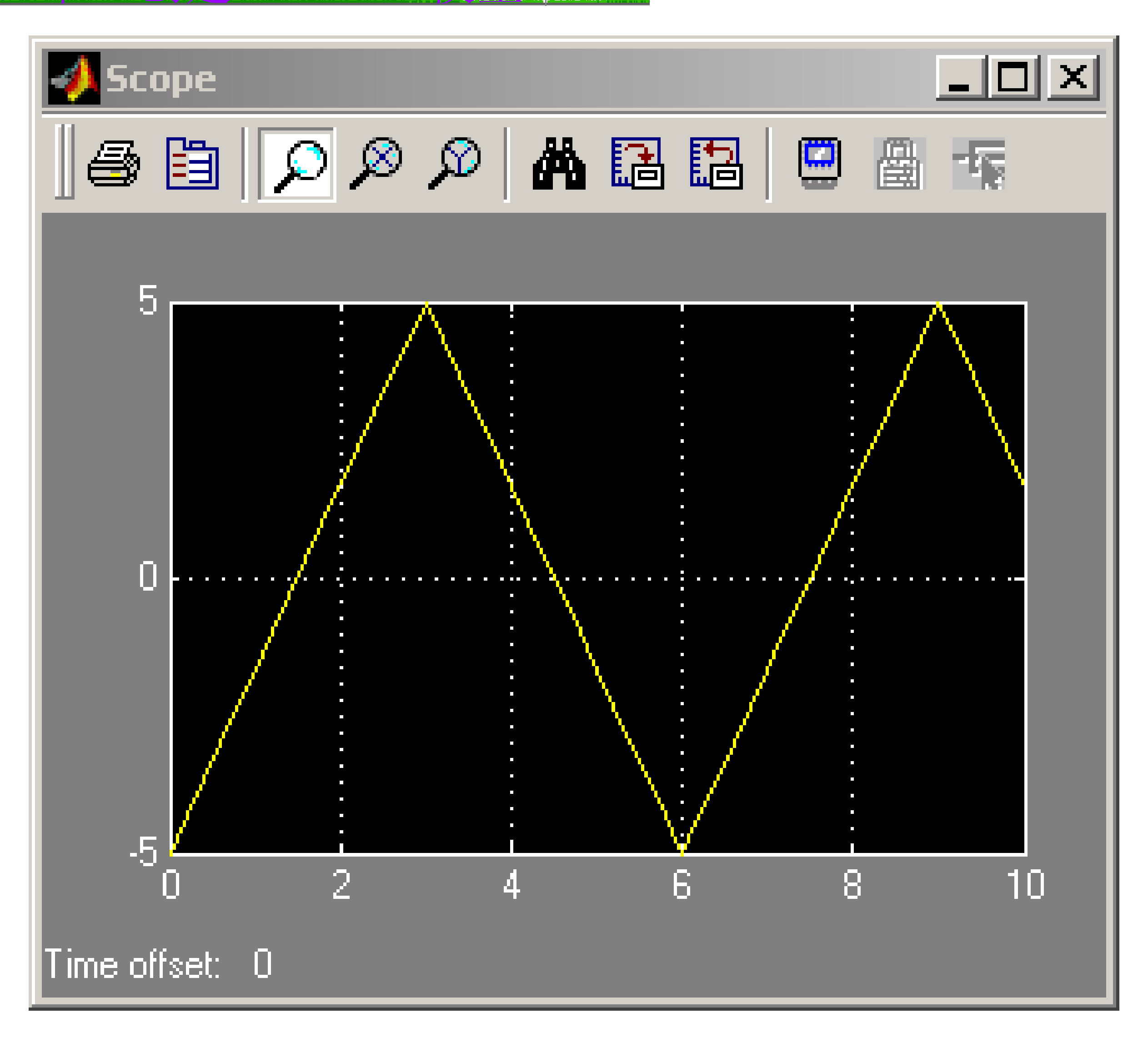


图4.36 三角波输入的仿真结果

4.5.4 向量与矩阵

在前面的系统模型中，Simulink所使用的信号均是标量。其实，Simulink 也能够传递和使用向量信号。例如，向量增益可以作用在一个标量信号上，产生一个向量输出。在缺省情况下，模块对向量中的逐个元素进行操作，就像MATLAB中的数组运算一样，如图4.37所示。

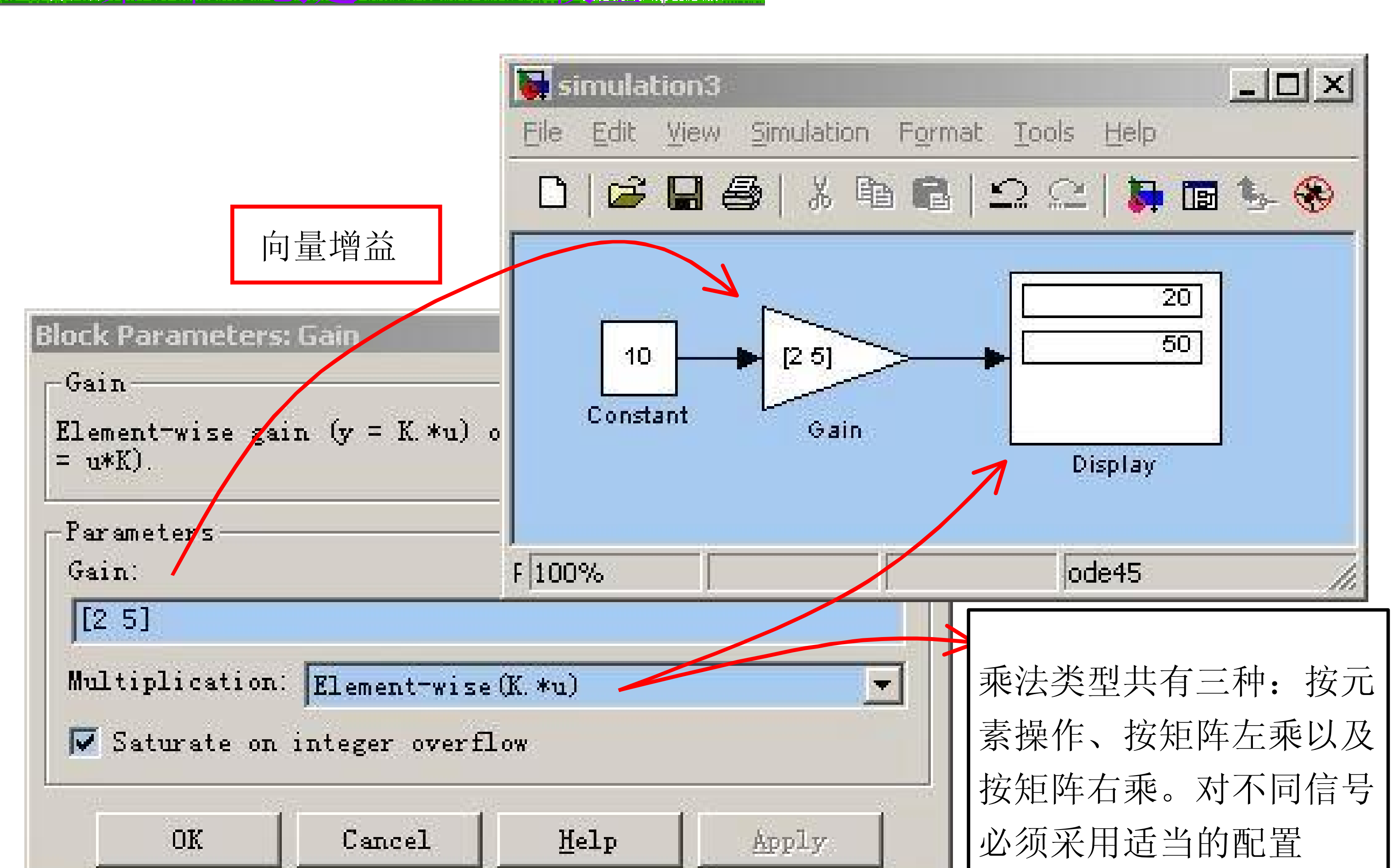


图4.37　向量增益示意图

Simulink 4最重要的特性就是支持矩阵形式的信号，它可以区分行和列向量并传递矩阵。通过对模块做适当的配置，可以使模块能够接受矩阵作为模块参数。在上面的例子中，如果Constant模块的参数为一矩阵，并且Gain增益模块被配置成按矩阵乘的定义从左边乘上输入向量，则Display 块能够感知到输入信号的尺寸，即1×2行向量，并对边框做适当调整，如图4.38所示。

simulation4

File Edit View Simulation Format Tools Help

Constant Gain Display

[2 5]* u

60 12

100% ode45

Block Parameters: Gain

Gain

Element-wise gain (y = K.* ... = u*K).

Parameters

Gain:

[2 5]

Multiplication: Matrix (K*u)

Saturate on integer overflow

OK Cancel Help Apply

图4.38 矩阵输入与向量增益示意图

4.5.5 MATLAB Function与Function模块

除了使用上述的方式进行Simulink与MATLAB之间的数据交互，用户还可以使用Functions and Tables 模块库中的 Function模块（简称为Fcn模块）或Functions and Tables 模块库中的MATLAB Function模块（简称为MATLAB Fcn模块）进行彼此间的数据交互。

Fcn模块一般用来实现简单的函数关系，在Fcn模块中：

(1) 输入总是表示成u，u可以是一个向量。

(2) 可以使用 C 语言表达式，例如sin(u[1])+cos(u[2])。

(3) 输出永远为一个标量。

MATLAB Fcn一般用来调用MATLAB函数来实现一定的功能，在MATLAB Fcn模块中：

(1) 所要调用的函数只能有一个输出（可以是一个向量）。

(2) 单输入函数只需使用函数名，多输入函数输入需要引用相应的元素，如mean、sqrt、myfunc(u(1),u(2))。

(3) 在每个仿真步长内都需要调用MATLAB解释器。

使用Fcn模块与MATLAB Fcn模块进行Simulink与MATLAB之间的数据交互如图4.39所示。

函数表达式

MATLAB函数

图4.39 使用Fcn与MATLAB Fcn模块进行数据交互



4.6 使用Simulink进行简单的仿真

【例4.1】 信号平方运算。系统的功能是对输入的信号进行平方运算。现要求建立系统的Simulink模型并进行简单的仿真分析。具体要求如下：

(1) 系统输入信号源：幅值为 2 的正弦波。

(2) 使用Scope 显示原始信号和结果信号。

(3) 生成系统运算部分的子系统

(4) 生成信号标签并传递。

解： 首先选择系统所需的如下模块（组件）：

(1)　Sources 库中的Sine Wave块。

(2)　Math 库中的Product块。

(3)　Signals and Systems库中的Mux块。

(4)　Sinks 库中的Scope块。

然后进行如下的操作：

(1) 连接系统模块。

(2) 选择一个包含Product和Mux块的区域，建立相应的子系统。

(3) 在主系统中生成输入信号的标签，在子系统中生成输出信号的标签。

(4) 传递信号的标签。

(5) 改变输入和输出端口的名字。

(6) 保存模型。

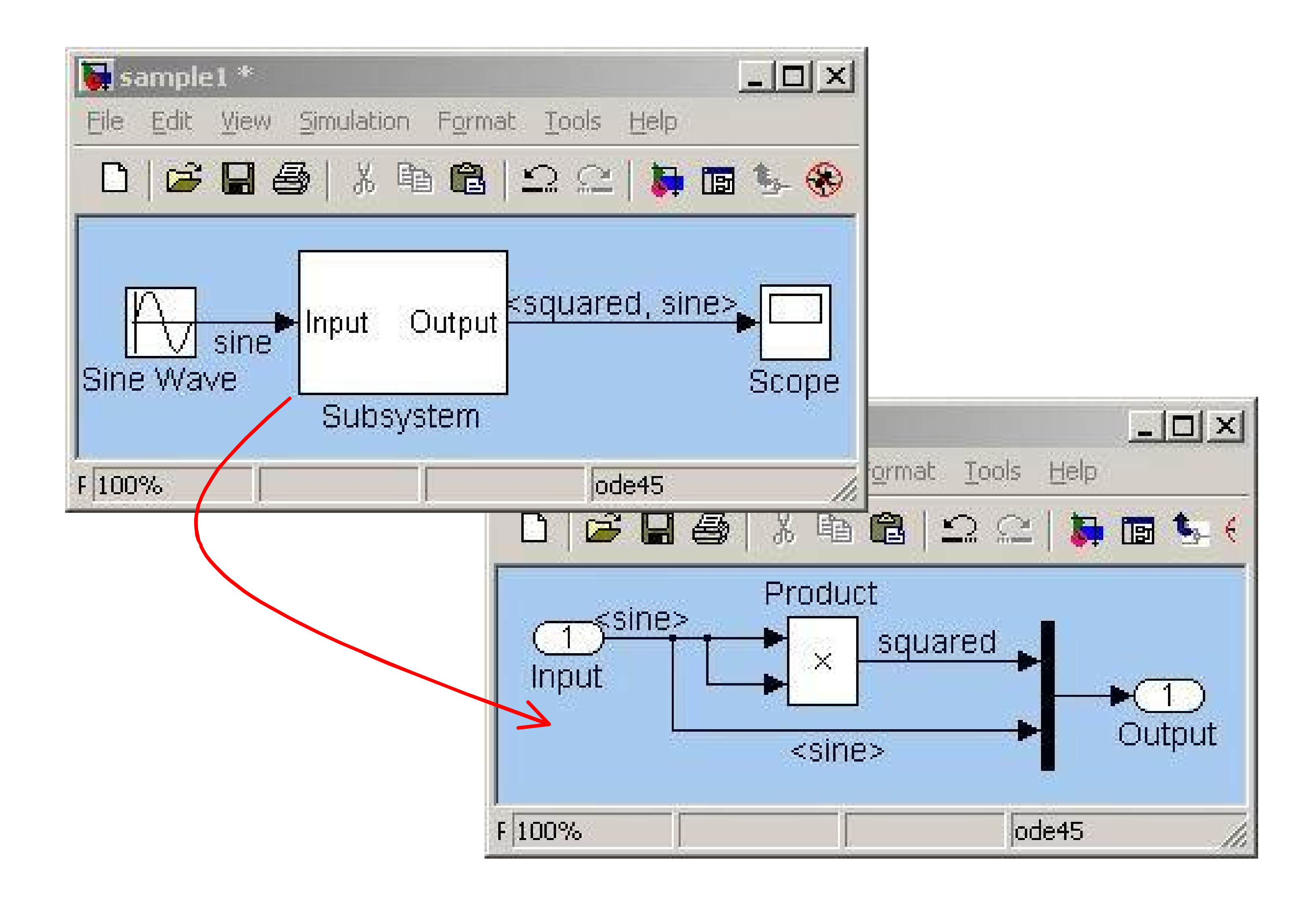


图4.40 平方运算系统模型

最后，进行模块参数设置并使用默认的仿真参数进行仿真。在本例中，只需要对系统输入信号源Sine Wave模块进行参数设置即可（双击Sine Wave模块），设置正弦信号的幅值为2，如图4.41所示。系统仿真结果如图4.42所示。

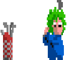




图4.41 Sine Wave模块参数设置

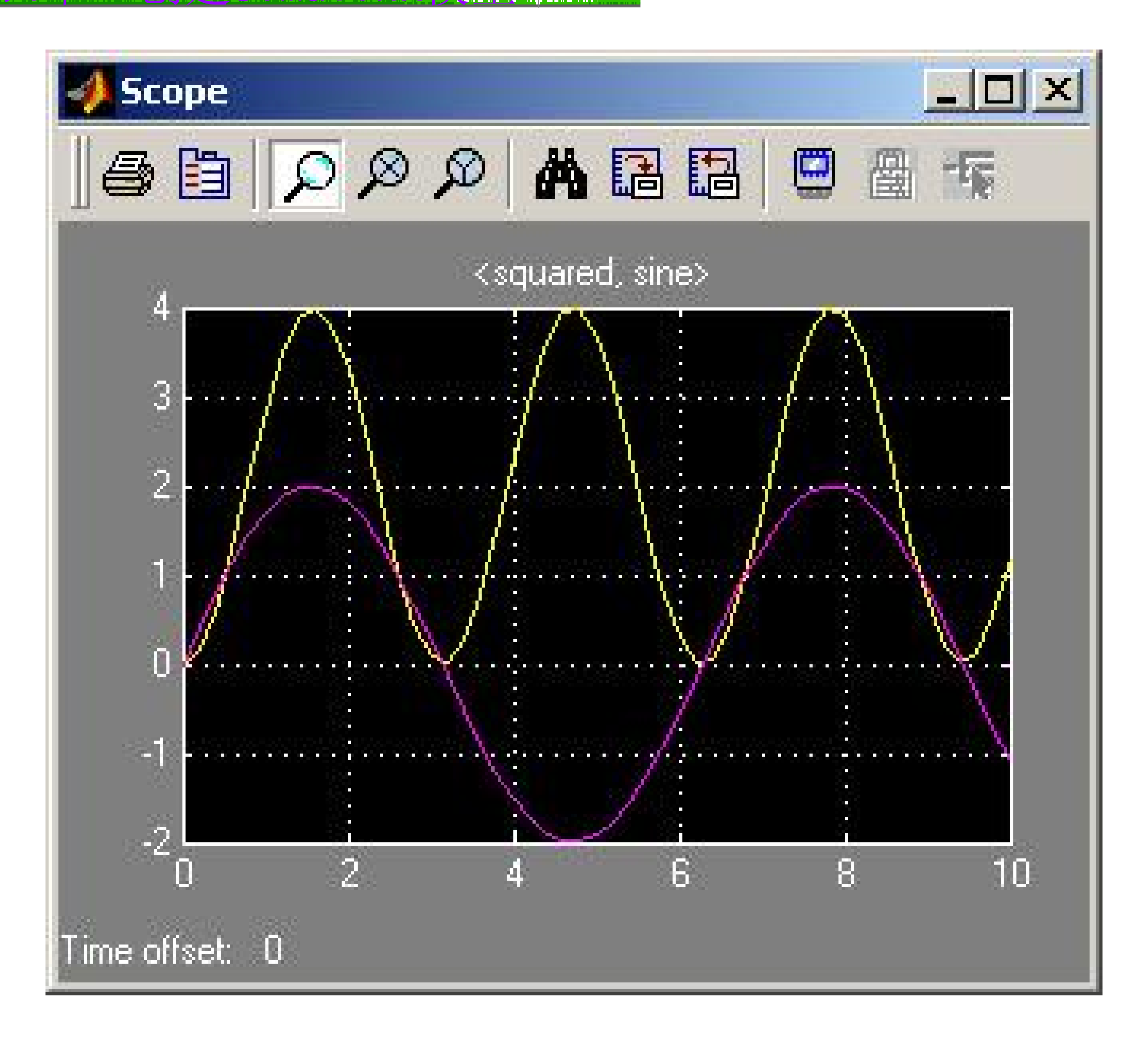


图4.42 系统仿真结果

本章对Simulink的模型构建作了一个比较全面的介绍，对于不同领域的工程技术人员，都可以利用本章介绍的基本知识对大部分的系统进行建模与简单分析。至此，用户应该能够熟练快速地建立自己的系统模型。Simulink的功能非常强大，它可以大大提高系统设计、仿真与分析的效率。本章是Simulink最基础的知识。后面的章节将详细介绍使用Simulink进行系统仿真的高级技术，以及Simulink系统仿真的原理。



第5章 动态系统的Simulink



5.1 简单系统的仿真分析

5.1.1 建立系统模型

首先根据系统的数学描述选择合适的Simulink系统模块，然后按照第4章中的方法建立此简单系统的系统模型。这里所使用的系统模块主要有：

(1) Sources模块库中的Sine Wave模块：用来作为系统的输入信号。

(2) Math模块库中的Relational Operator模块：用来实现系统中的时间逻辑关系。

(3) Sources模块库中的Clock模块：用来表示系统运行时间。

(4) Nonlinear模块库中的Switch模块：用来实现系统的输出选择。

(5) Math模块库中的Gain模块：用来实现系统中的信号增益。

图5.1所示为此简单系统的系统模型。

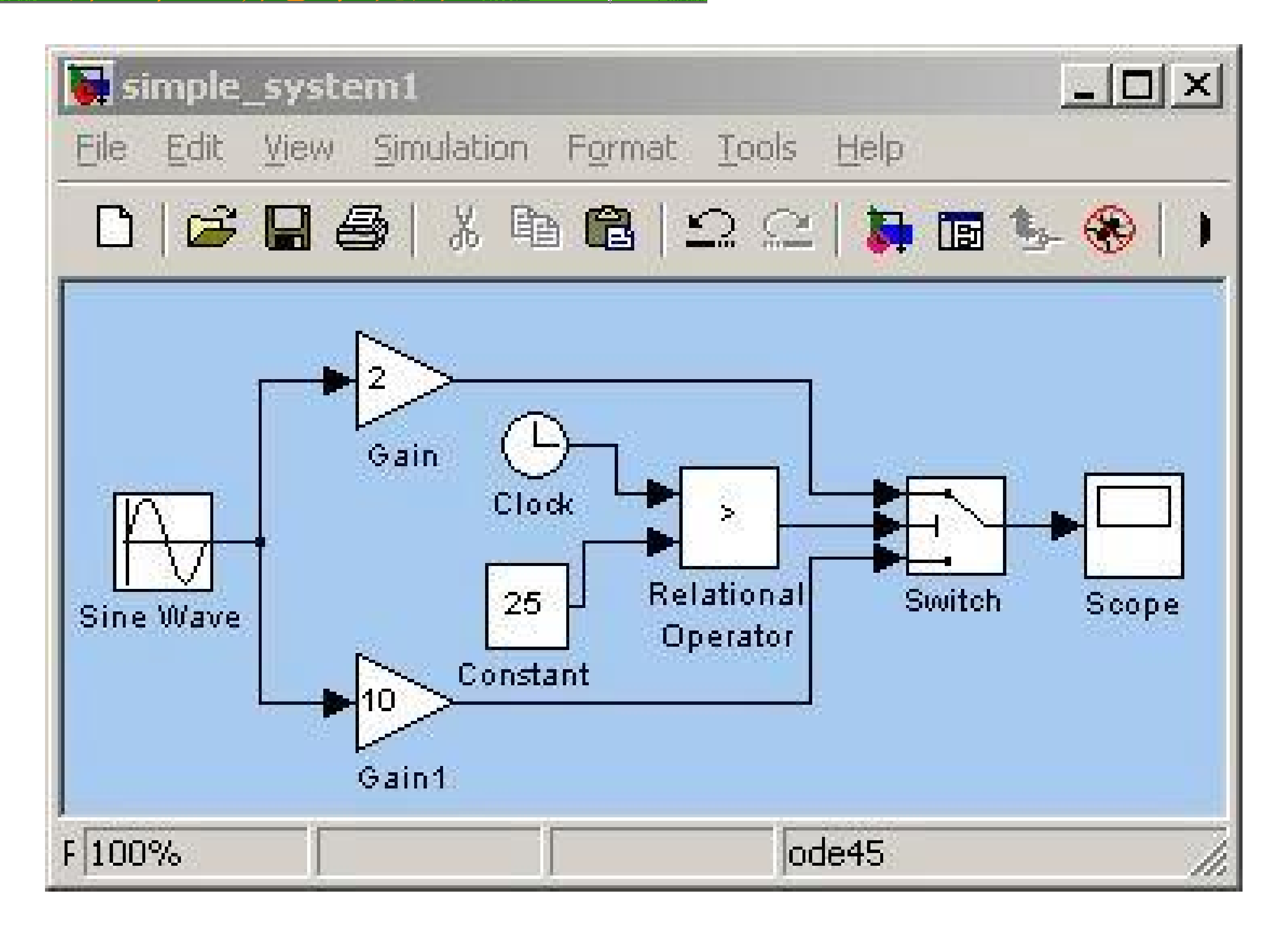

simple_system1
File Edit View Simulation Format Tools Help
2
Gain
Clock
Sine Wave
25
Constant
Relational Operator
Switch
Scope
10
Gain1
100%
ode45

图5.1 简单系统模型

5.1.2 系统模块参数设置

在完成系统模型的建立之后，需要对系统中各模块的参数进行合理的设置。这里采用的模块参数设置如下所述：

(1) Sine Wave模块：采用Simulink默认的参数设置，即单位幅值、单位频率的正弦信号。

(2) Relational Operator模块：其参数设置为“>”，如图5.2所示。

(3) Clock模块：采用默认参数设置，如图5.3所示。

(4) Switch模块：设定Switch模块的Threshold值为0.5（其实只要大于0小于1即可，因为Switch模块在输入端口2的输入大于或等于给定的阈值Threshold时，模块输出为第一端口的输入，否则为第三端口的输入），从而实现此系统的输出随仿真时间进行正确的切换。如图5.4所示。

(5) Gain模块：其参数设置如图5.1系统模型中所示，这里不再赘述。

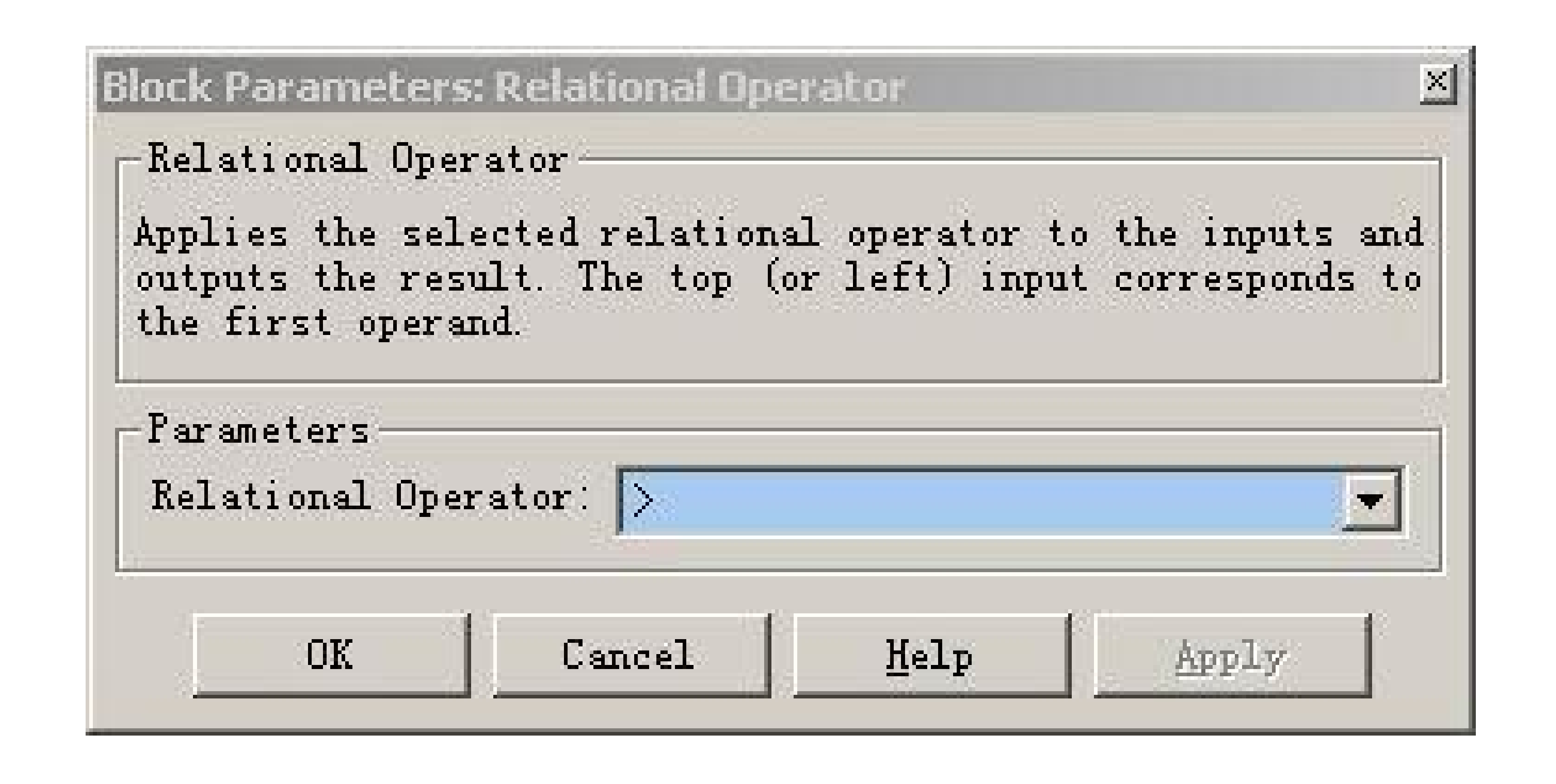


图5.2 Relational Operator模块参数设置

Block Parameters: Switch

Switch

Pass through input 1 when input 2 is greater than or equal to threshold; otherwise, pass through input 3. The inputs are numbered top to bottom (or left to right).

Parameters

Threshold:

0.5

OK　Cancel　Help　Apply

阈值大于0小于1

图5.3 Clock模块参数设置

Block Parameters: Clock

Clock

Output the current simulation time.

Parameters

Display time

Decimation:

10

OK　Cancel　Help　Apply

图5.4 Switch模块参数设置

5.1.3 系统仿真参数设置及仿真分析

在对系统模型中各个模块进行正确且合适的参数设置之后，需要对系统仿真参数进行必要的设置以开始仿真。

在缺省情况下，Simulink默认的仿真起始时间为0 s，仿真结束时间为10 s。对于此简单系统，当时间大于25时 系统输出才开始转换，因此需要设置合适的仿真时间。设置仿真时间的方法为：选择菜单Simulation中的Simulation Parameters（或使用快捷键Ctrl+E），打开仿真参数设置对话框，在Solver选项卡中设置系统仿真时间区间。设置系统仿真起始时间为0 s、结束时间为100 s，如图5.5所示。

在系统模块参数与系统仿真参数设置完毕之后，用户便可开始系统仿真了。运行仿真的方法有如下几种：

(1) 选择菜单Simulation中的Start Simulation。

(2) 使用系统组合热键Ctrl+T。

(3) 使用模型编辑器工具栏中的Play按钮（即黑色三角形）。

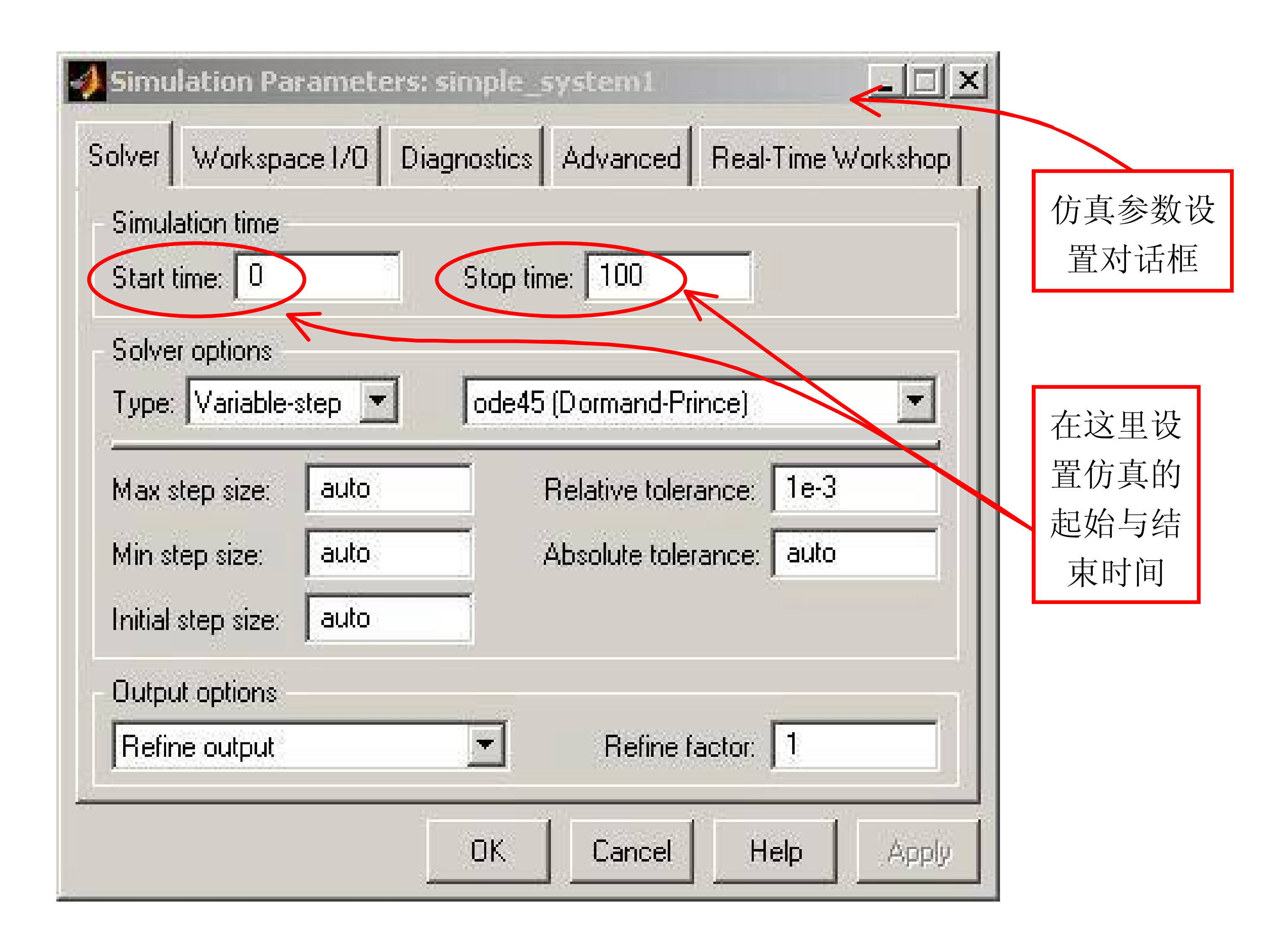


图5.5 系统仿真时间设置

当系统仿真结束后，双击系统模型中的Scope模块，显示的系统仿真结果如图5.6所示。从图5.6中可以看出，系统仿真输出曲线非常不平滑；而对此系统的数学描述进行分析可知，系统输出应该为光滑曲线。这是由于在仿真过程中没有设置合适的仿真步长，而是使用Simulink的默认仿真步长设置所造成的。因此，对动态系统的仿真步长需要进行合适的设置。

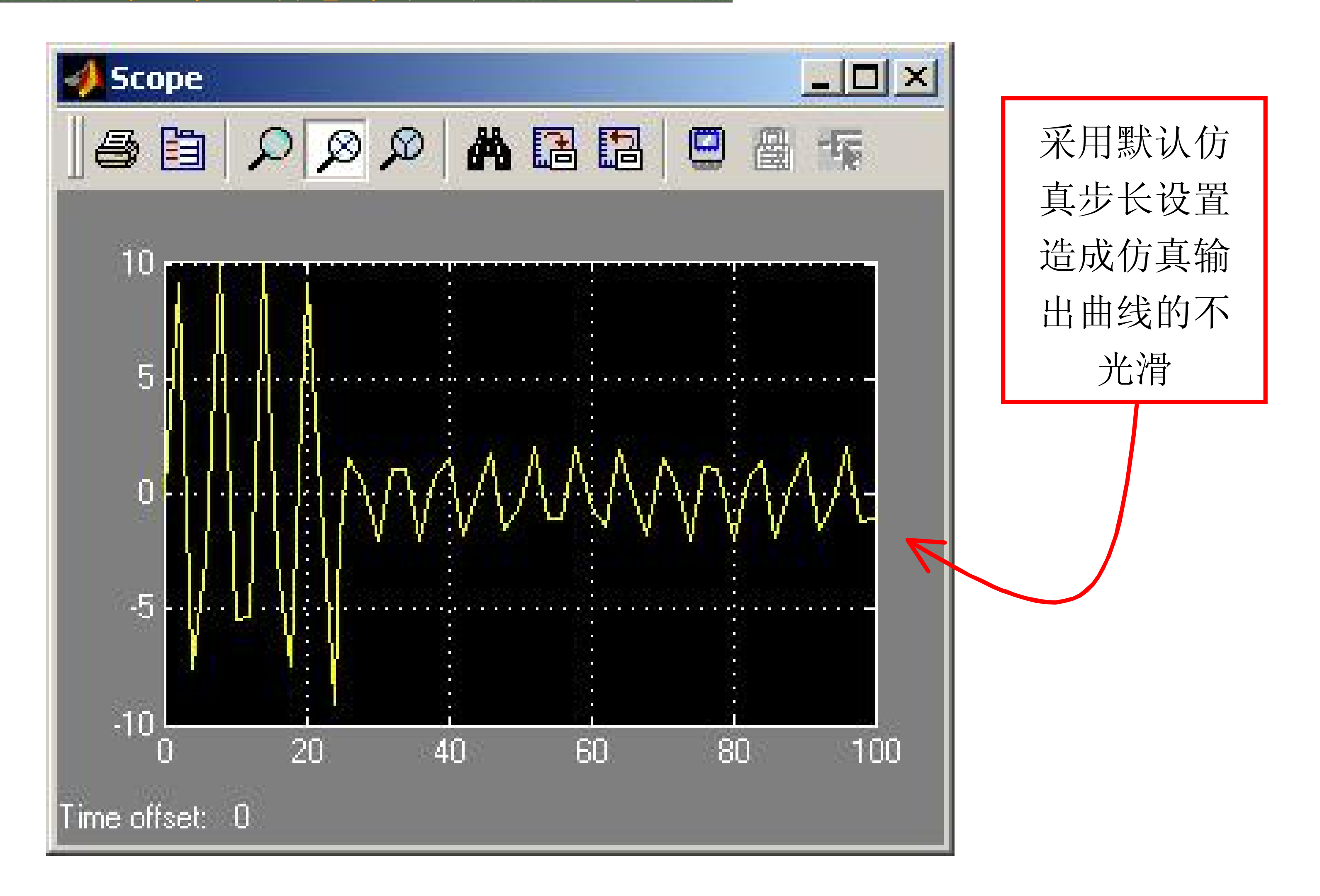


图5.6 系统仿真结果输出曲线

5.1.4仿真步长设置

仿真参数的选择对仿真结果有很大的影响。对于简单系统，由于系统中并不存在状态变量，因此每一次计算都应该是准确的（不考虑数据截断误差）。在使用Simulink对简单系统进行仿真时，影响仿真结果输出的因素有仿真起始时间、结束时间和仿真步长。对于简单系统仿真来说，不管采用何种求解器，Simulink总是在仿真过程中选用最大的仿真步长。

如果仿真时间区间较长，而且最大步长设置采用默认取值auto，则会导致系统在仿真时使用大的步长，因为Simulink的仿真步长是通过下式得到的：

$$h = \frac{t_{final} - t_{start}}{50}$$

Simulation Parameters: simple_system1

Solver | Workspace I/O | Diagnostics | Advanced | Real-Time Workshop

Simulation time

Start time: 0 Stop time: 100

Solver options

Type: Variable-step ode45 (Dormand-Prince)

Max step size: 0.1 Relative tolerance: 1e-3

Min step size: auto Absolute tolerance: auto

Initial step size: auto

Output options

Refine output Refine factor: 1

OK Cancel Help Apply

最大仿真步长设置为0.1

图5.7 系统最大仿真步长设置

仿真输出曲线变得比较光滑，仿真结果得到明显改善

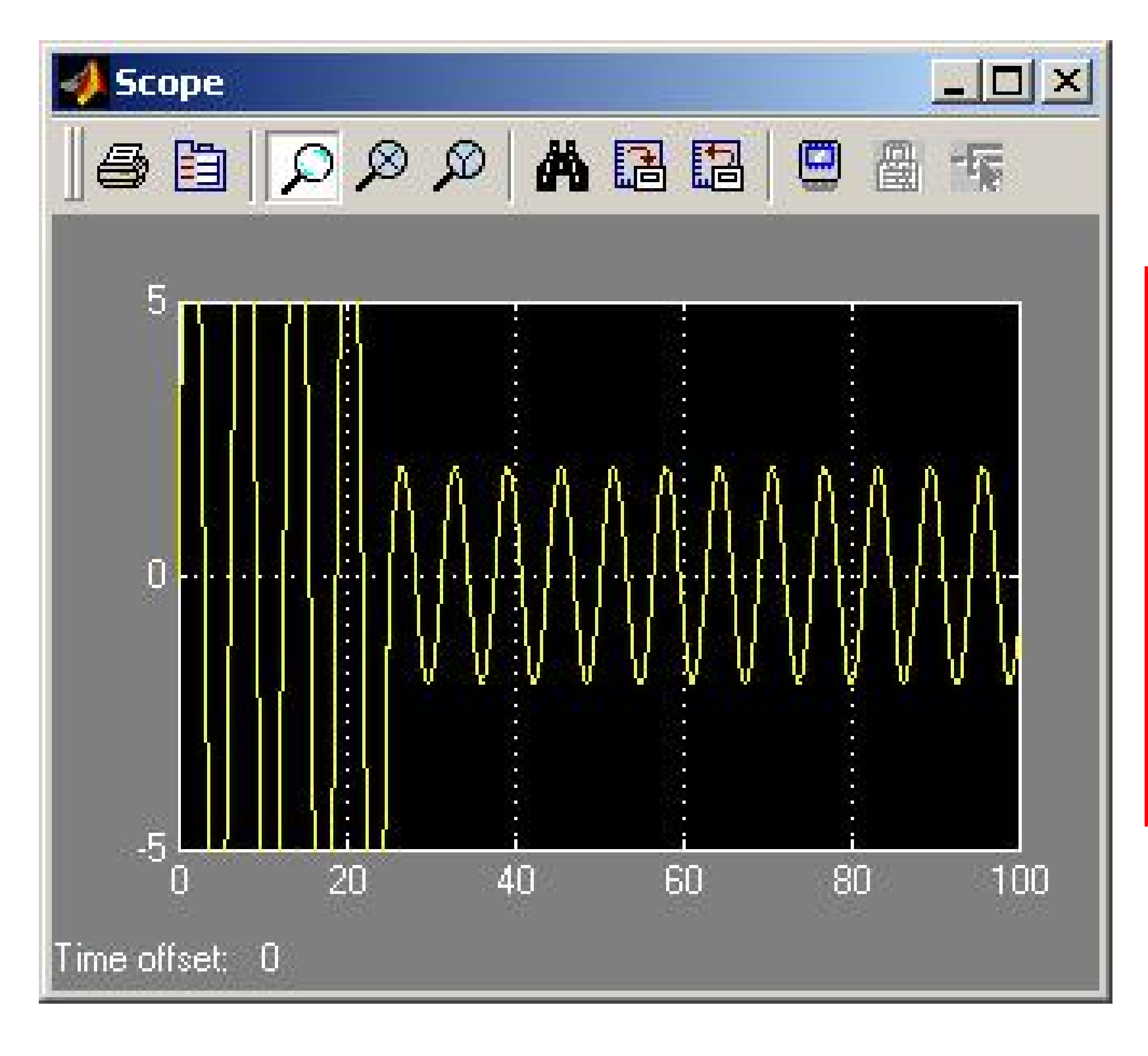
图5.8 系统最大仿真步长为0.1下的仿真输出结果



5.2 Scope高级使用技术

5.2.1 Scope模块的使用

这里以5.1节中简单系统仿真输出结果为例说明Scope模块的使用技术。图5.9所示为默认设置下此简单系统仿真结果输出显示。（5.1节中对Scope显示的动态范围进行了调整，因此与图5.9中显示的不一样。）



默认设置下Scope模块所显示的系统仿真结果输出（Scope模块动态范围不足以显示仿真输出结果）

图5.9 默认设置下Scope模块的显示

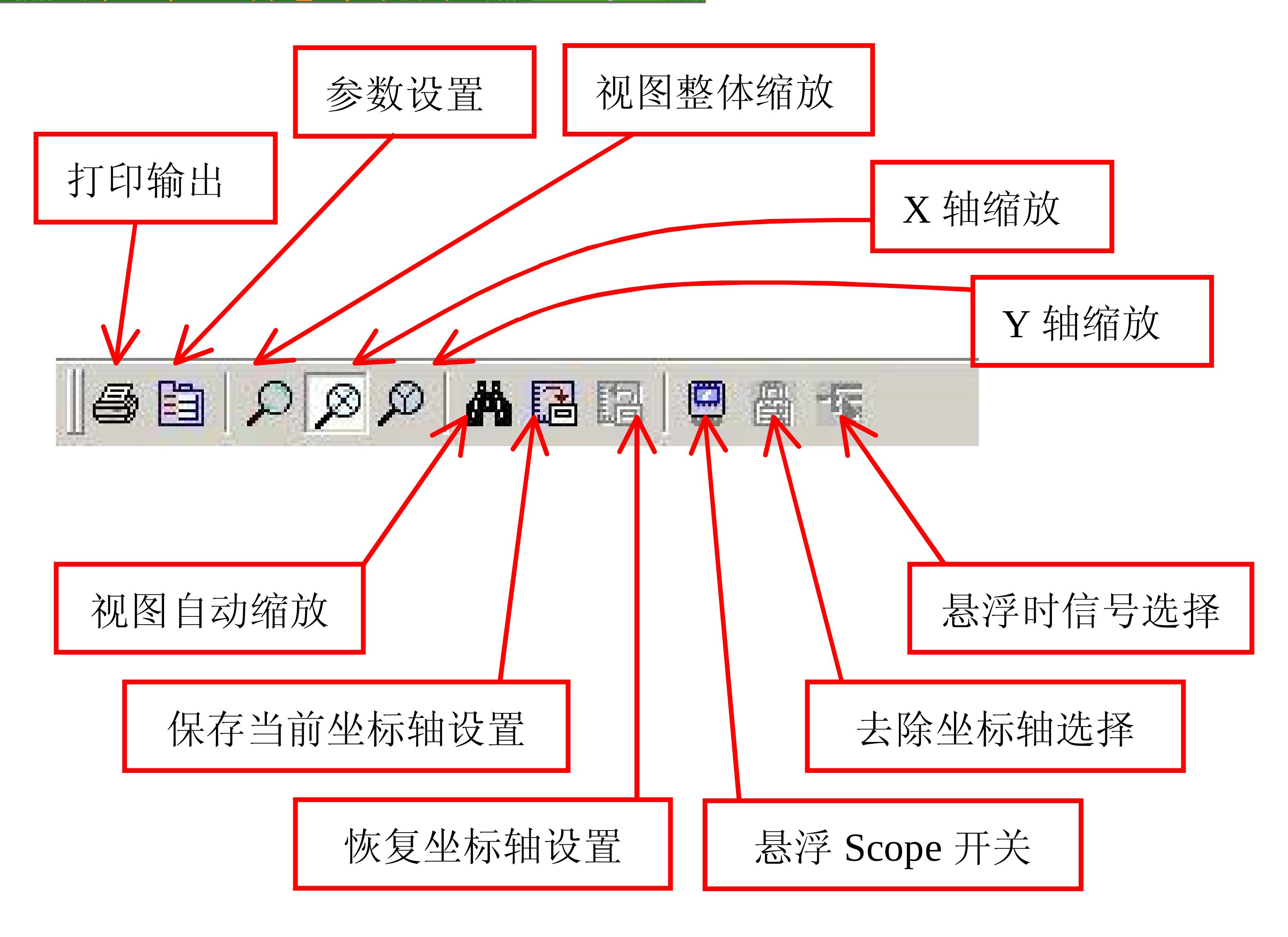


图5.10 Scope模块工具栏按钮命令

下面分别对各项功能进行详细介绍。

1) 打印输出（Print）

将系统仿真结果的输出信号打印出来。

2) 视图自动缩放（Autoscale）

Simulink自动调整显示范围以匹配系统仿真输出信号的动态范围。在图5.9中采用默认设置，如果自动缩放视图，则可以获得更好的显示效果，如图5.11所示。

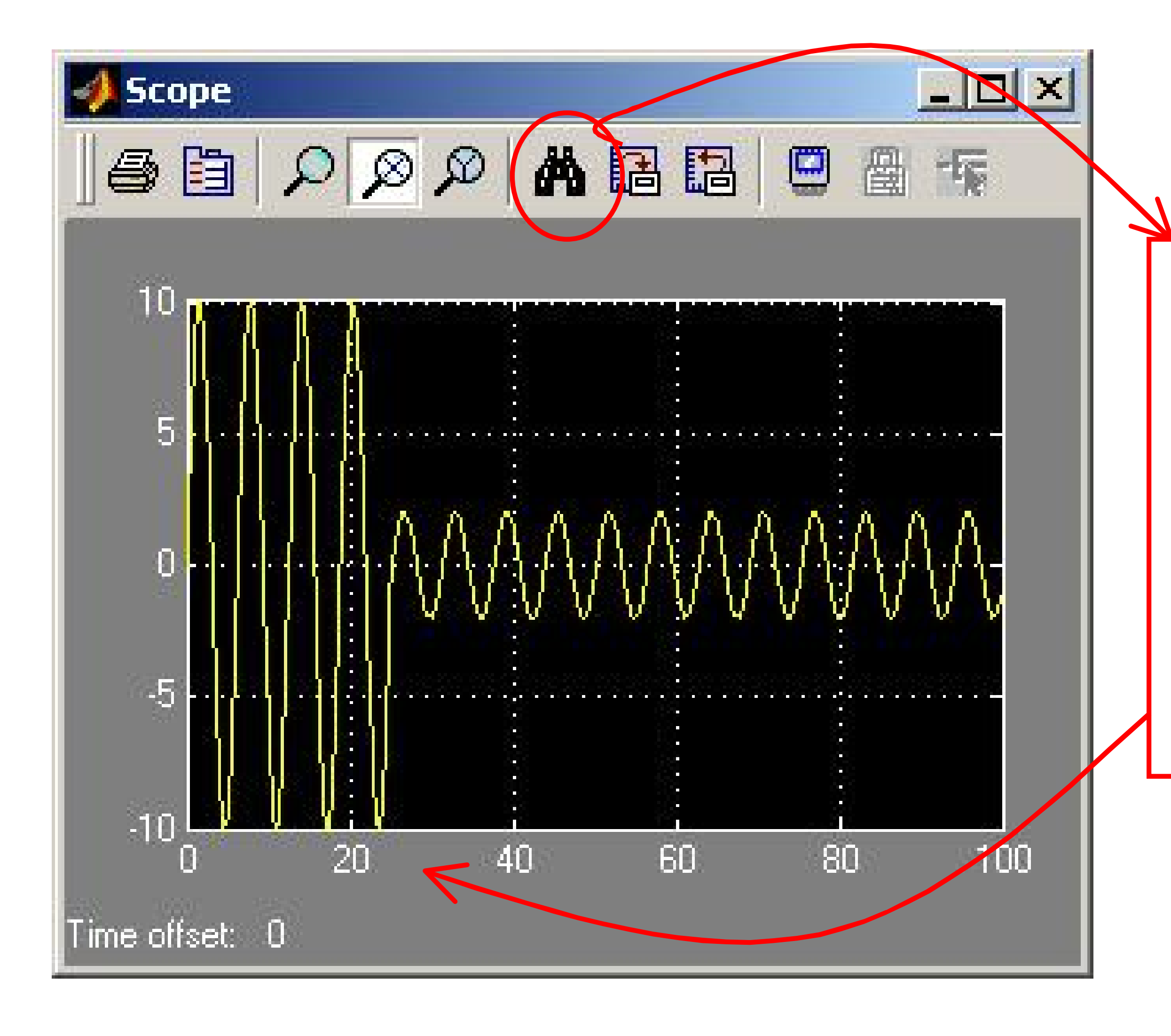

Scope
10
5
0
-5
-10
0
20
40
60
80
100
Time offset: 0
按下此按钮则可以使信号动态范围与Scope显示相匹配，从而方便用户对输出信号进行观察

图5.11 视图自动缩放

3) X轴缩放、Y轴缩放以及视图整体缩放

对信号的指定范围进行缩放，可以分别对X坐标轴、Y坐标轴或同时对X、Y坐标轴（即整体视图）的信号显示作缩放，以满足用户对信号做局部观察的需要。首先单击缩放按钮，然后选择需要观察的信号范围即可，如图5.12所示。如果用户需要缩小视图，单击鼠标右键，选择弹出菜单的Zoom out即可。

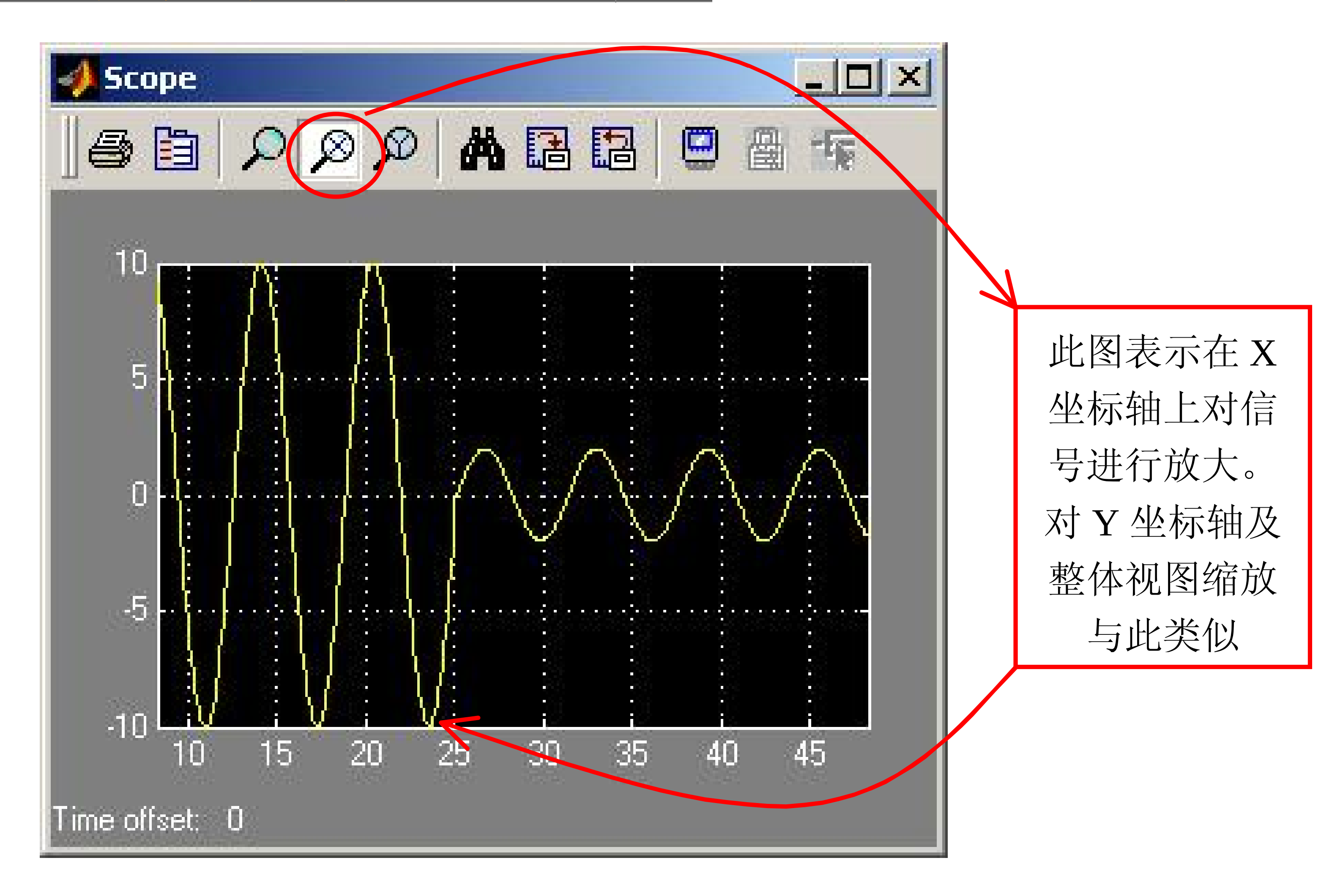

Scope
10
5
0
-5
-10
10 15 20 25 30 35 40 45
Time offset: 0
此图表示在 X 坐标轴上对信号进行放大。对 Y 坐标轴及整体视图缩放与此类似

图5.12 视图缩放

4) 保存与恢复坐标轴设置

在使用Scope模块观测输出信号时，用户可以保存坐标轴设置。这样，当信号的视图发生改变后，单击恢复坐标轴设置可以恢复以前保存的坐标轴设置，如图5.13所示。

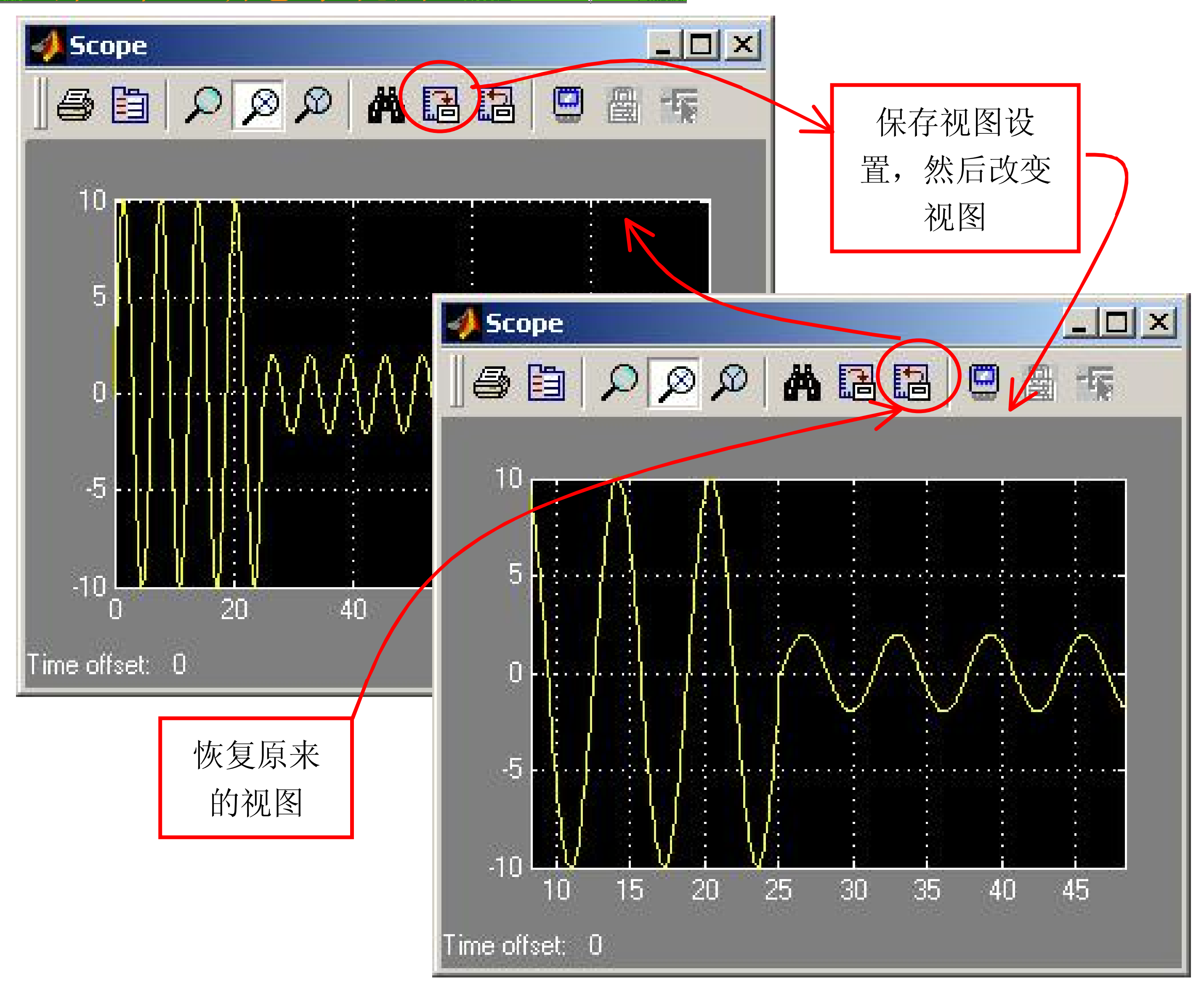


图5.13 保存与恢复视图设置

5) Scope的参数设置

使用Scope模块的参数设置选项卡能够对系统仿真输出结果显示进行更多的控制，而不仅仅是上述的简单控制。图5.14、图5.15所示分别为Scope模块参数设置选项卡中的General选项卡与Data History选项卡。

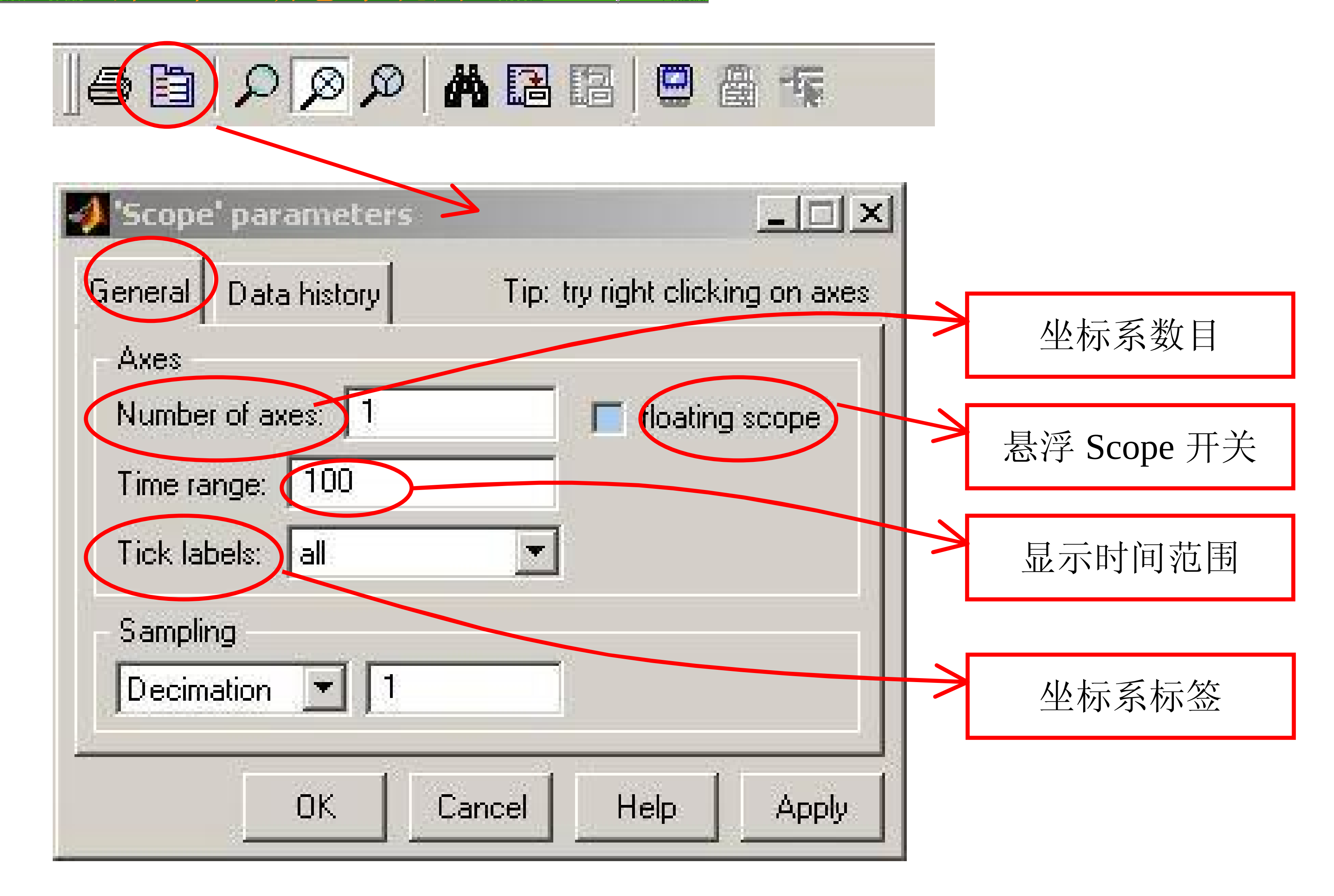


图5.14 Scope模块的General选项卡

信号显示点数限制

保存信号至工作空间变量

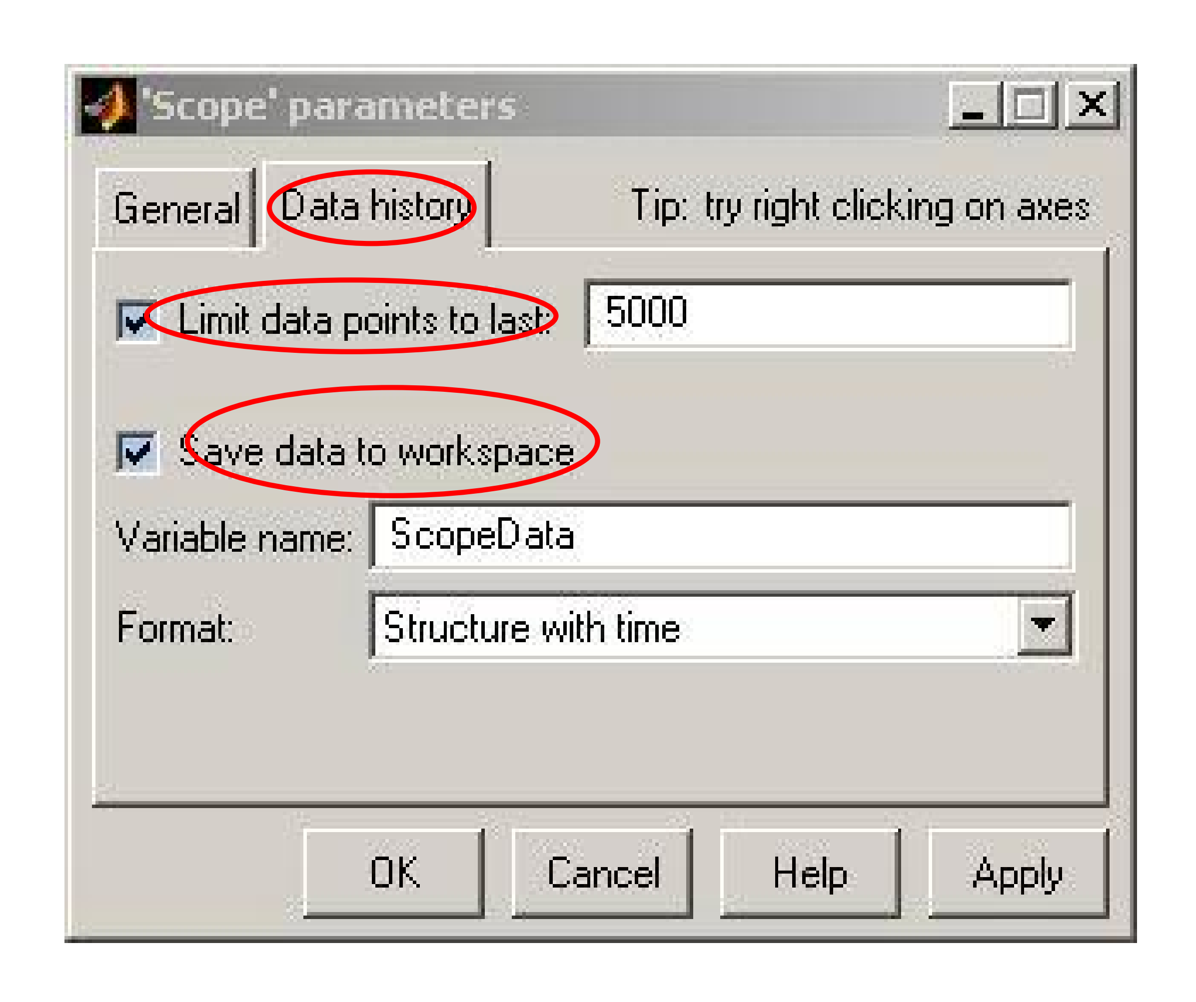


图5.15 Scope模块的Data history选项卡

下面简单介绍一下各选项卡的功能与使用。

1) 坐标系数目（Number of axes）

功能描述：在一个Scope输出模块中使用多个坐标系窗口同时输出多个信号。在默认设置下，Scope模块仅显示一个坐标系窗口。

2) 悬浮Scope开关（Floating scope）

功能描述：将Scope模块切换为悬浮Scope模块。悬浮Scope模块将在5.2.3节中进行介绍。

3) 显示时间范围（Time range）

功能描述：设置信号显示的时间范围。注意：信号显示的时间范围与系统仿真时间范围并不等同，并且坐标系所示的时间范围并非为绝对时间，而是指相对时间范围，坐标系的左下角的时间偏移（Time offset）给出了时间的起始偏移量（即显示时间范围的起始时刻）。

4) 坐标系标签（Tick labels）

功能描述：确定Scope模块中各坐标系是否带有坐标轴标签。此选项提供了三种选择：全部坐标系都使用坐标轴标签（all）、最

下方坐标系使用标签（bottom axis only）以及都不使用标签（none）。用户最好使用标签，这有利于对信号的观察理解.

5) 信号显示点数限制（Limit data points to last）

功能描述：限制信号显示的数据点的数目，Scope模块会自动对信号进行截取以显示信号的最后n个点（这里n为设置的数值）。

6) 保存信号至工作空间变量（Save data to workspace）

功能描述：将由Scope模块显示的信号保存到Matlab工作空间变量中，以便于对信号进行更多的定量分析。数据保存类型有三种：带时间变量的结构体（structure with time）、结构体（structure）以及数组变量（Array）。这与前面所介绍的Sinks模块库中的To workspace模块类似。

此外，在Scope模块中的坐标系中单击鼠标右键，选择弹出菜单中坐标系属性设置命令（axes properties），将弹出图5.16所示的坐标系属性设置对话框。用户可以对Scope模块的坐标系标题与显示信号范围进行合适的设置，以满足仿真输出结果显示的需要。



图5.16 坐标系属性设置对话框

5.2.2 Display模块的使用

在某些情况下，用户需要观察或动态显示某个信号的数值结果时，可以选用Display模块，它既可以显示单个信号，也可以显示向量信号或矩阵信号（帧信号）。当信号的显示范围超出了Display模块的边界，会在Display模块的右下角出现一个向下的三角，表示还有信号的值没被显示出来，这时用户只需用鼠标拉大Display模块的显示面板即可。

5.2.3 悬浮Scope模块

在系统仿真分析中，用户往往需要对多个输出信号进行观察分析。如果将每一个信号都与一个Scope模块相连接，则系统模型中必定会存在多个Scope模块，使得系统模型不够简练，而且难以对不同Scope模块中显示的信号进行直观的比较。Sinks模块库中Floating Scope模块（悬浮Scope模块）可以很好地解决这一问题。

1. 悬浮Scope模块的使用方法

使用悬浮Scope模块的方法有如下两种：

(1) 直接将Sinks模块库中的Floating Scope模块拖动到指定的系统模型之中。然后选择需要显示的信号并进行适当的设置，最后进行系统仿真并显示系统中指定的信号。

(2) 设置普通的Scope模块为Floating Scope模块。用户只需选择图5.14中所示的悬浮Scope开关即可。其后的操作与（1）一致。

例如，对于图5.17所示的动态系统模型，使用Floating Scope与悬浮Display模块显示指定的信号。

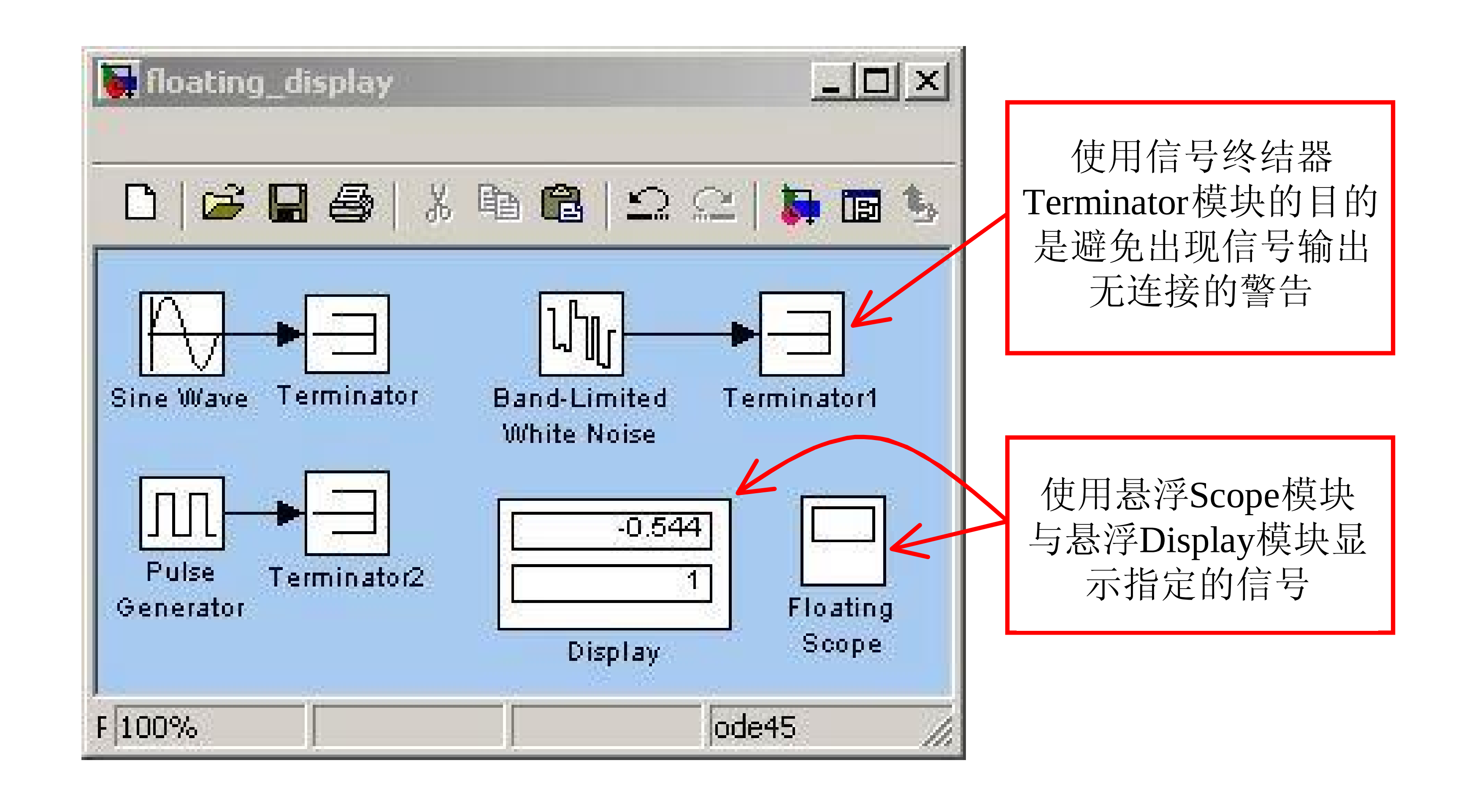


图5.17　使用悬浮Scope模块与悬浮Display模块

2. 使用悬浮Scope模块显示信号的参数设置

对于图5.17所示的系统模型，要使用悬浮Scope模块显示指定的信号，必须进行正确的设置。

1) 设置需要显示的信号

使用悬浮Scope模块的信号选择器选择需要显示的信号：首先打开信号选择器对话框，然后在可显示信号列表中选择需要显示的信号，这里选择显示正弦信号与方波信号。信号选择如图5.18所示。

2) 设置信号存储缓冲区与全局变量

在缺省情况下，Simulink重复使用存储信号的缓存区。也就是说，Simulink信号都是局部变量。使用悬浮Scope模块显示指定信号，由于信号与模块之间没有实际的连接，因此局部变量不再适用。故用户应当避免Simulink 对变量的缓存区重复使用，需要对其进行正确设置。

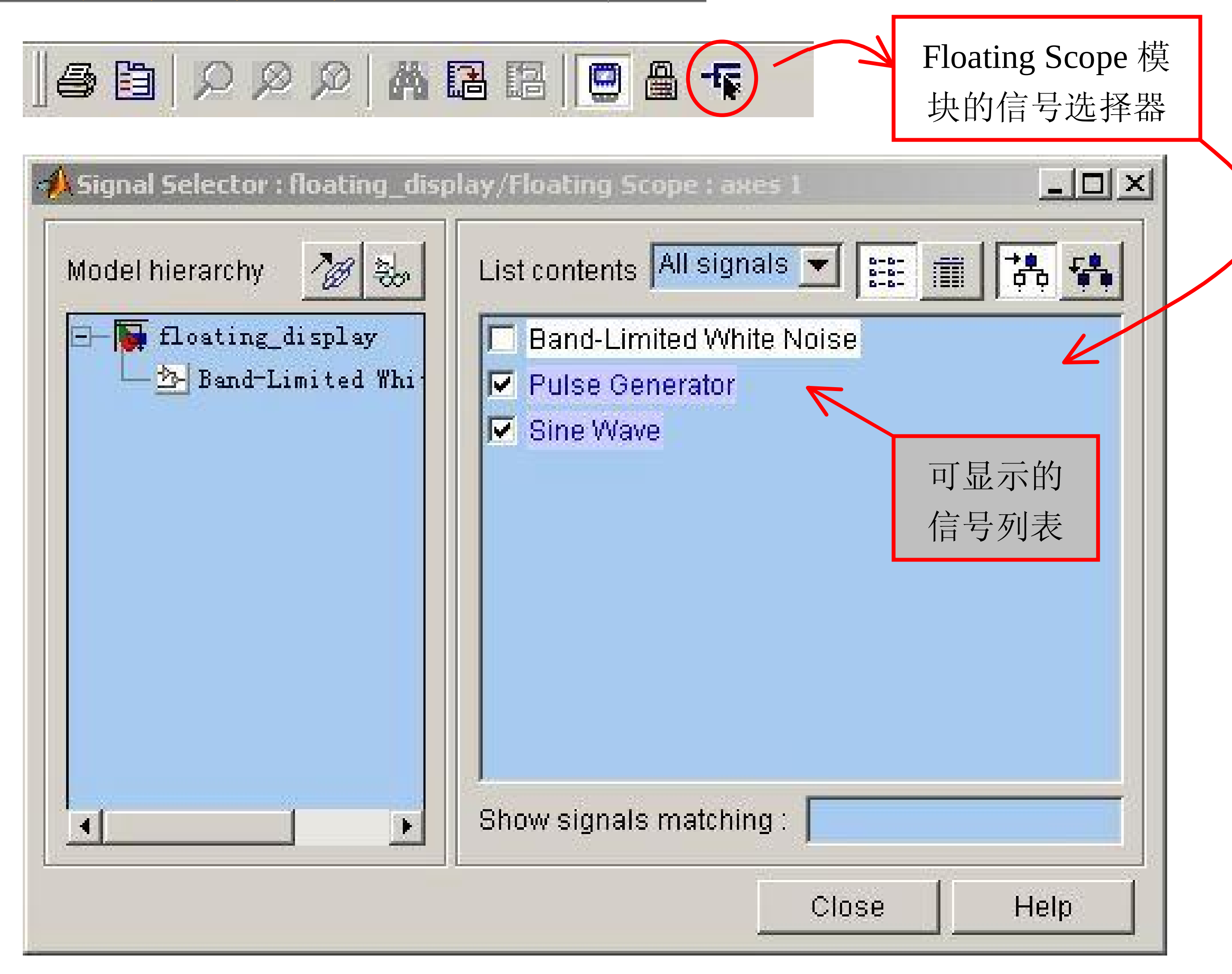


图5.18 悬浮Scope模块的信号选择

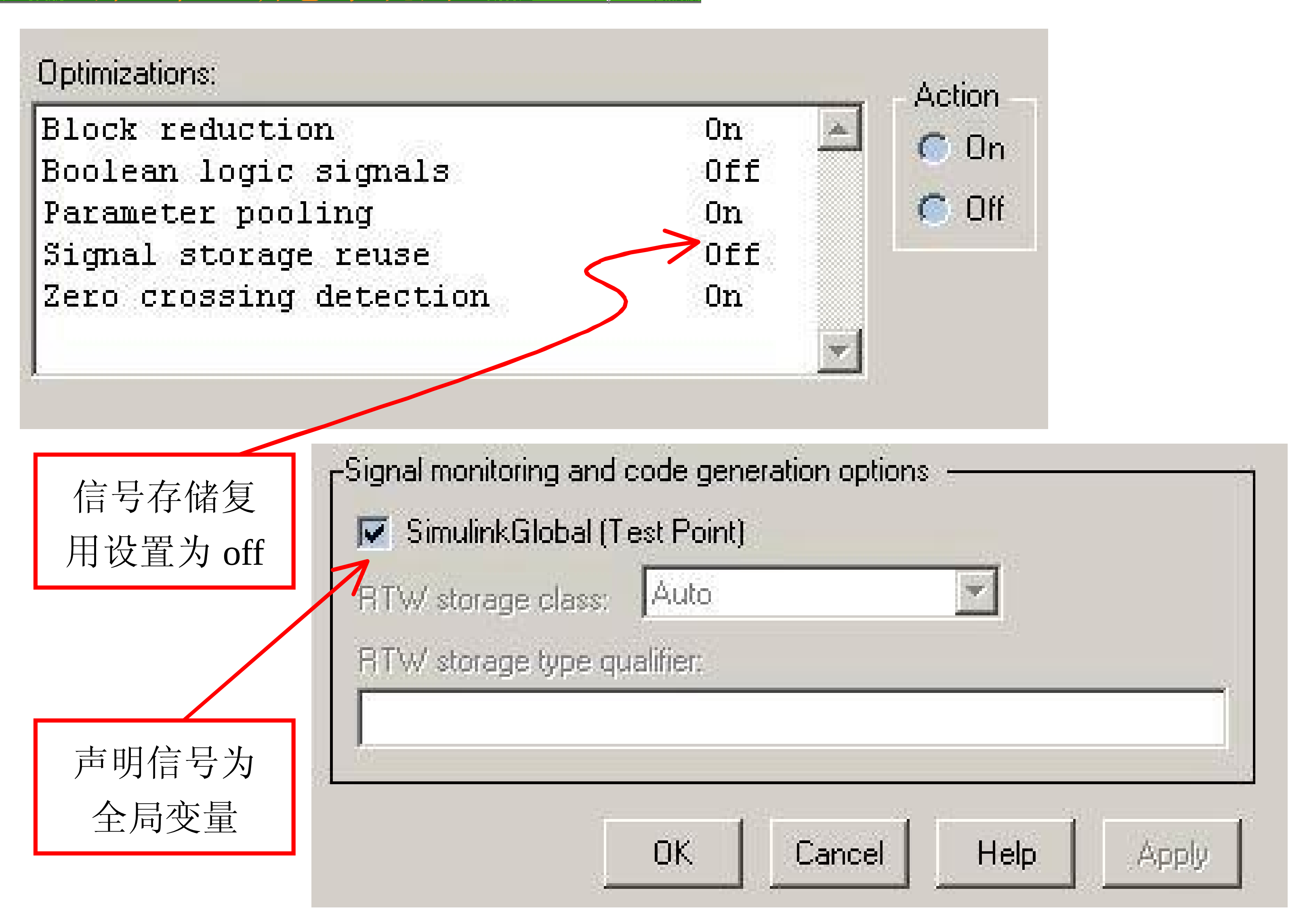


图5.19 信号存储缓冲区设置

3. 运行系统仿真

在相应的参数设置完成之后，运行系统仿真，系统仿真输出结果如图5.20所示。从图中可以看出，正弦信号与方波信号被正确显示出来。

图5.20　系统仿真结果（悬浮Scope模块输出）



5.3 离散系统的仿真分析

5.3.1 人口变化系统的数学模型

这是一个简单的人口变化模型。在此模型中，设某一年的人口数目为，其中表示年份，它与上一年的人口、人口繁殖速率以及新增资源所能满足的个体数目之间的动力学方程由如下的差分方程所描述：

$$p(n) = rp(n-1)\left[1 - \frac{p(n-1)}{K}\right]$$

从此差分方程中可以看出，此人口变化系统为一非线性离散系统。如果设人口初始值、人口繁殖速率、新增资源所能满足的个体数目，要求建立此人口动态变化系统的系统模型，并分析人口数目在0至100年之间的变化趋势。

5.3.2 建立人口变化系统的模型

在建立此人口变化的非线性离散系统模型之前，首先对离散系统模块库（Discrete模块库）中比较常用的模块作简单的介绍。

(1) Unit Delay模块：其主要功能是将输入信号延迟一个采样时间，它是离散系统的差分方程描述以及离散系统仿真的基础。在仿真时只要设置延迟模块的初始值便可计算系统输出。

(2) Zero-Order Hold模块：其主要功能是对信号进行零阶保持。

使用Simulink对离散系统进行仿真时，单位延迟是由Discrete模块库中的Unit Delay模块来完成的。对于人口变化系统模型而言，需要将作为Unit Delay模块的输入以得到，然后按照系统的差分方程来建立人口变化系统的模型。

图5.21　人口变化系统模型

5.3.3 系统模块参数设置

系统模型建立之后，首先需要按照系统的要求设置各个模块的参数，如下所述：

(1) 增益模块Gain表示人口繁殖速率，故取值为1.05。

(2) 模块Gain1表示新增资源所能满足的个体数目，故取值为1000000。

(3) Unit Delay模块参数设置。对于离散系统而言，必须正确设置所有离散模块的初始取值，否则系统仿真结果会出现错误。这是因为在不同的初始值下，系统的稳定性会发生变化。单位延迟模块的参数设置如图5.22所示。

Block Parameters: Unit Delay

Unit Delay

Sample and hold with one sample period delay.

Parameters

Initial conditions:

100000

Sample time (-1 for inherited):

1

OK　Cancel　Help　Apply

初始条件相当于人口的初始值 $p(0)$，故取值为 100000

图5.22 单位延迟模块的参数设置

5.3.4系统仿真参数设置及仿真分析

在正确设置系统模型中各模块的模块参数之后，需要对系统仿真参数进行设置。下面介绍离散系统的仿真参数设置，在此之前首先介绍系统仿真的基本原理。这可以使用户加深对离散系统仿真的理解，并且更好的掌握离散系统仿真技术。



图5.23 系统的仿真原理

下面设置人口变化系统的仿真参数：

(1) 仿真时间设置：按照系统仿真的要求，设置系统仿真时间范围为0~100。

(2) 离散求解器与仿真步长设置：对离散系统进行仿真需要使用离散求解器。对于离散系统的仿真，无论是采用定步长求解器还是采用变步长求解器，都可以对离散系统进行精确的求解。这里选择定步长求解器对此系统进行仿真分析。至于定步长与变步长的区别将在后面专门进行介绍。

使用Simulation菜单中的Simulation Parameters设置系统仿真参数，如图5.24所示。

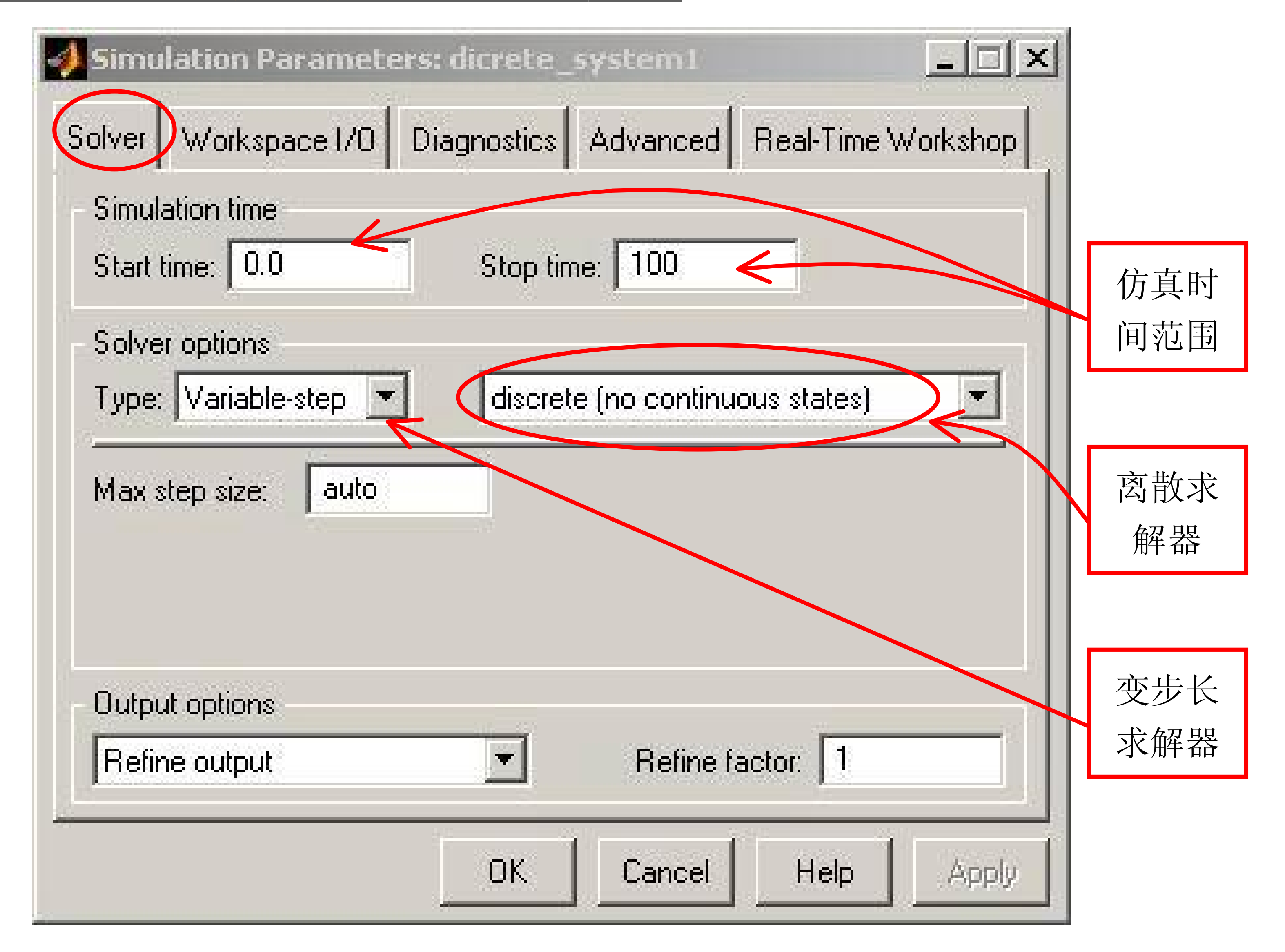


图5.24 系统仿真参数设置

在对系统中各模块参数以及系统仿真参数进行正确设置之后，运行系统仿真，对人口数目在指定的时间范围之内的变化趋势进行分析。图5.25所示为系统仿真输出结果。

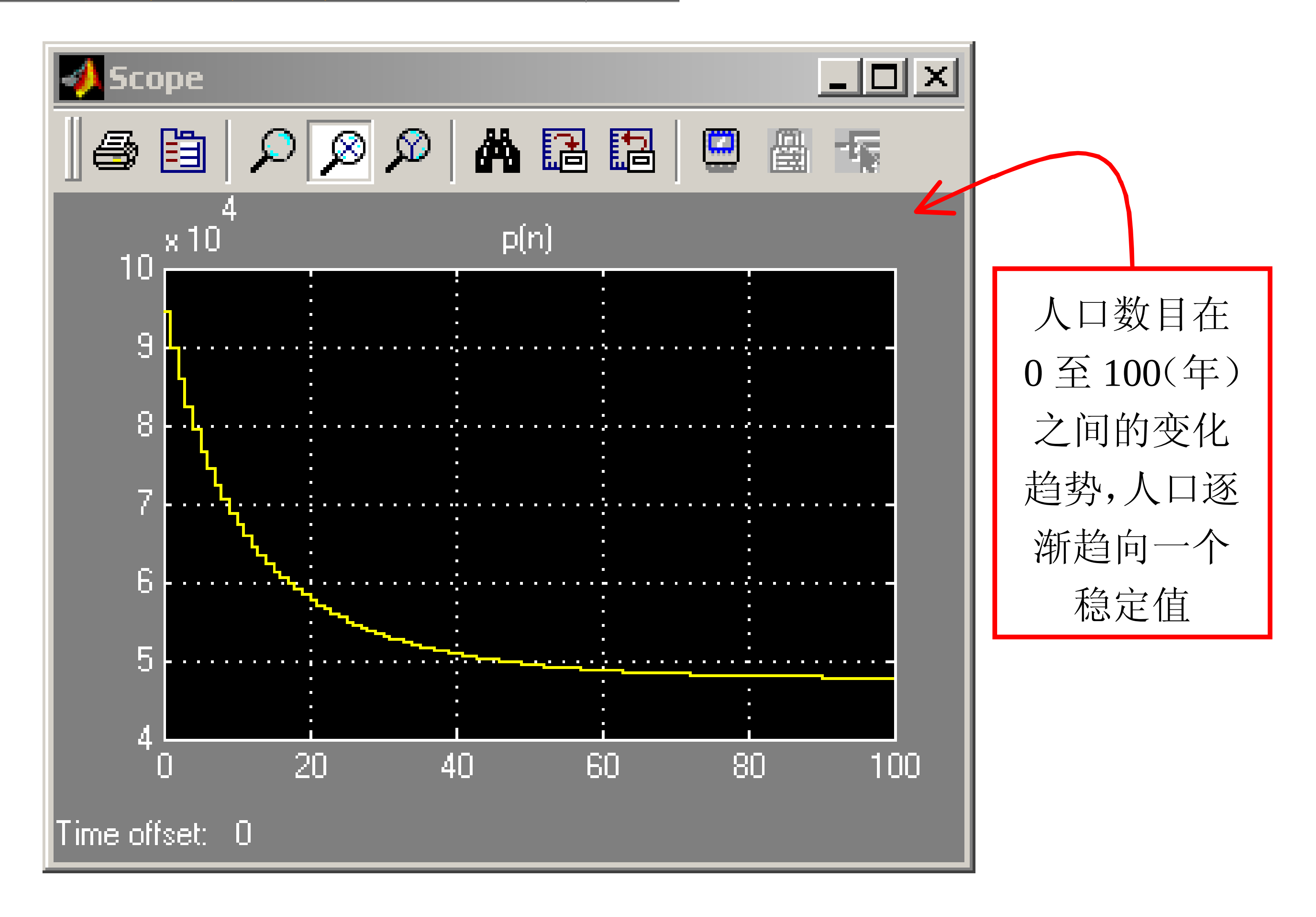


图5.25 人口变化系统仿真结果

5.3.5 定步长仿真与变步长仿真

对于一个单速率离散系统的仿真，选择定步长求解器对仿真来说已经足够了。但是在对多速率离散系统进行仿真时，采用变步长仿真则具有更多的优势。这里所谓的单速率与多速率离散系统，是指离散系统中各系统模块采样时间是否一致，如果所有模块的采用时间均相同，则此系统为单速率离散系统，否则为多速率离散系统。一般而言，使用变步长求解器对离散系统进行仿真，其效率要优于定步长求解器，如图5.26所示。

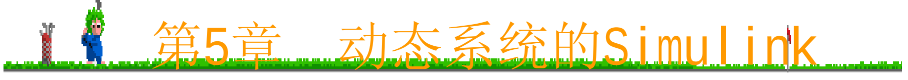


图5.26 定步长求解器与变步长求解器的仿真时刻对比

图5.26表示某一离散系统模型中的某两个系统模块具有不同的采样时间，模块1的采样时间为0.4s，而模块2的采样时间为0.6 s。图中每个黑色粗实线代表了模块的一次更新，也就是模块的每一次采样。如果使用定步长求解器对此离散系统进行仿真，由于模块之间采样时间不同，所以步长选择要足够小以匹配所有的采样时刻；于是导致在某些时刻系统进行了不必要的计算（图中由椭圆曲线标志）。如果使用变步长求解器，仿真步长自动调整使其恰好匹配两个模块的更新，从而提高了系统仿真的效率。



5.4 连续系统的仿真分析

5.4.1 蹦极跳系统的数学模型

蹦极跳是一种挑战身体极限的运动，蹦极者系着一根弹力绳从高处的桥梁（或是山崖等）向下跳。在下落的过程中，蹦极者几乎是处于失重状态。按照牛顿运动规律，自由下落的物体的位置由下式确定：

$$m\ddot{x} = mg - a_1\dot{x} - a_2\left|\dot{x}\right|\dot{x}$$

其中为物体的质量，为重力加速度，为物体的位置，第二项与第三项表示空气的阻力。其中位置的基准为蹦极者开始跳下的位置（即选择桥梁作为位置的起点），低于桥梁的位置为正值，高于桥梁的位置为负值。如果物体系在一个弹性常数为的弹力绳索上，定义绳索下端的初始位置为，则其对落体位置的影响为

$$b(x)=\begin{cases}-kx, & x>0\\ 0, & x\le 0\end{cases}$$

因此整个蹦极跳系统的数学描述为

$$m\ddot{x}=mg+b(x)-a_1\dot{x}-a_2\left|\dot{x}\right|\dot{x}$$

5.4.2 建立蹦极跳系统的Simulink仿真模型

与建立离散系统模型类似，在建立蹦极跳系统的模型之前，首先对连续系统模块库Continuous中比较常用的模块简单的回顾。

(1) 积分器（Integrator）：积分器的主要功能在于对输入的连续信号进行积分运算。

(2) 微分器（Derivative）：微分器的主要功能在于对输入的连续信号进行微分运算。

在蹦极跳系统模型中，主要使用的系统模块有：

(1) Continuous模块库中的Integrator模块：用来实现系统中的微分运算。

(2) Functions & Tables模块库中的Fcn模块：用来实现系统中空气阻力的函数关系。

(3) Nonlinear模块库中的Switch模块：用来实现系统中弹力绳索的函数关系。

蹦极跳系统的模型框图如图5.27所示。

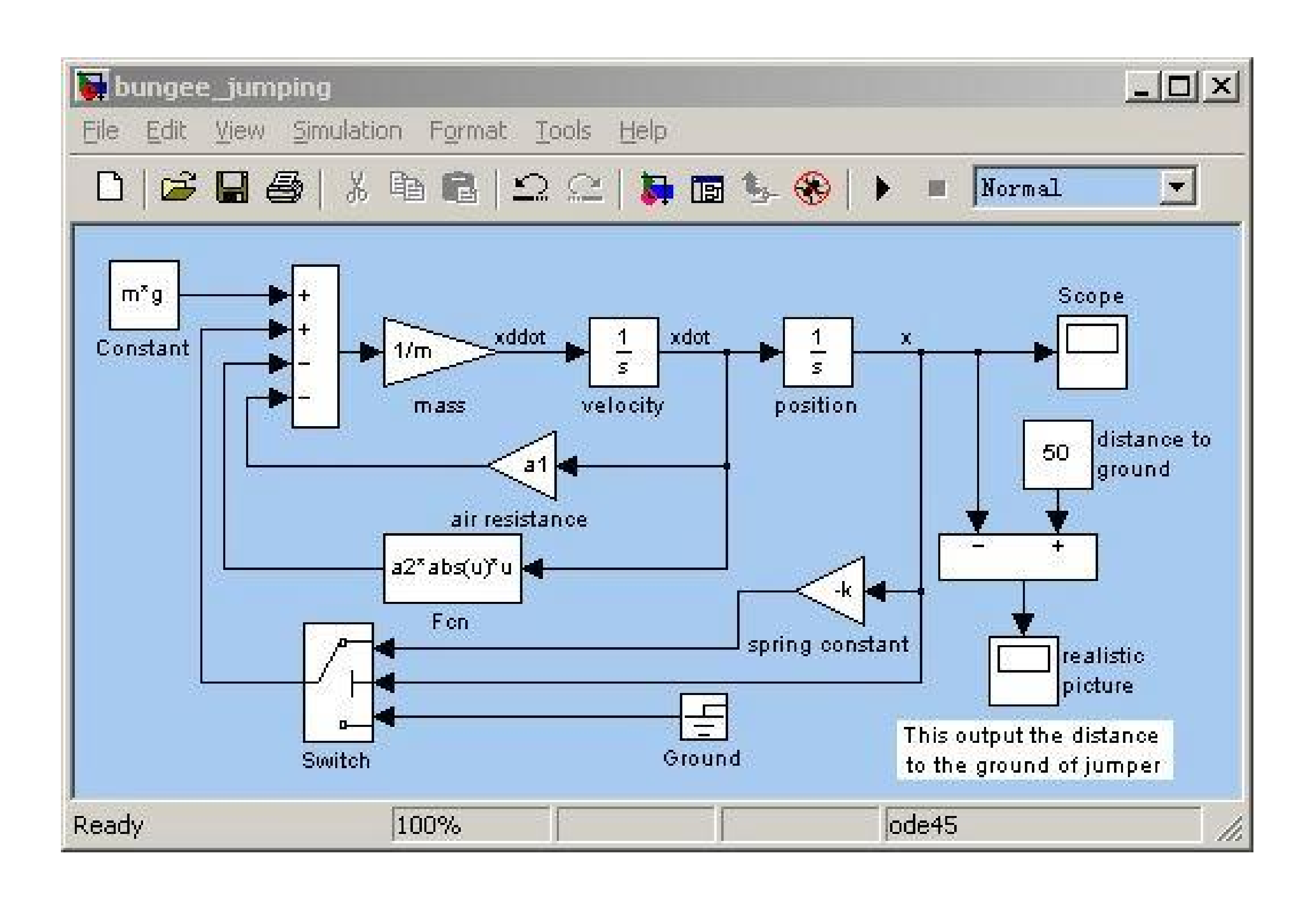


图5.27 蹦极跳系统模型

5.4.3 系统模块参数设置

在建立蹦极跳系统模型之后，需要设置系统模型中各个模块的参数。这里仅给出积分器模块velocity与position的参数设置，如图5.28所示。

在具有连续状态的连续系统中，千万不能忘记对积分器模块的初始值进行设置；因为在不同的初始值下，系统的动态规律可能大相径庭。至于其它模块的参数都比较简单，这里不再给出。

Block Parameters: velocity

Integrator

Continuous-time i

Parameters

External reset:

Initial conditio

Initial conditio

0

Limit output

Block Parameters: position

Integrator

Continuous-time integration of the input signal.

Parameters

External reset: none

Initial condition source: internal

Initial condition:

-30

图5.28 积分器模块velocity与position的参数设置

5.4.4 系统仿真参数设置与仿真分析

在对蹦极跳系统模型中各个模块的参数正确设置之后，需要设置系统仿真参数以对此系统进行仿真分析。在系统仿真参数设置之前，首先简单介绍一下Simulink的连续求解器。

5.3节中对离散系统的仿真原理做了简单的介绍，Simulink通过离散系统模型与Matlab求解器之间的交互完成离散系统的仿真；其实对于任何的动态系统，Simulink总是通过系统模型与Matlab求解器之间的交互来完成系统仿真。

微分方程的不同数值求解方法对应着不同的连续求解器。Simulink的连续求解器可以使用不同的数值求解方法对连续系统进行求解：

(1) 定步长连续求解器。可以使用如下的方法：ode5, ode4, ode3, ode2, ode1。

(2) 变步长连续求解器。可以使用如下的方法：ode45, ode23, ode113, ode15s, ode23s, ode23t， ode23tb。

使用Simulation菜单下的Simulation Parameters打开仿真参数设置对话框，对蹦极跳系统的仿真参数设置如下：

(1) 系统仿真时间范围为0~100 s。

(2) 其它仿真参数采用系统默认取值（变步长求解器、求解算法ode45、自动选择最大仿真步长、相对误差为1e-3）。

然后进行系统仿真。仿真输出结果如图5.29所示（蹦极者相对于地面的距离）。

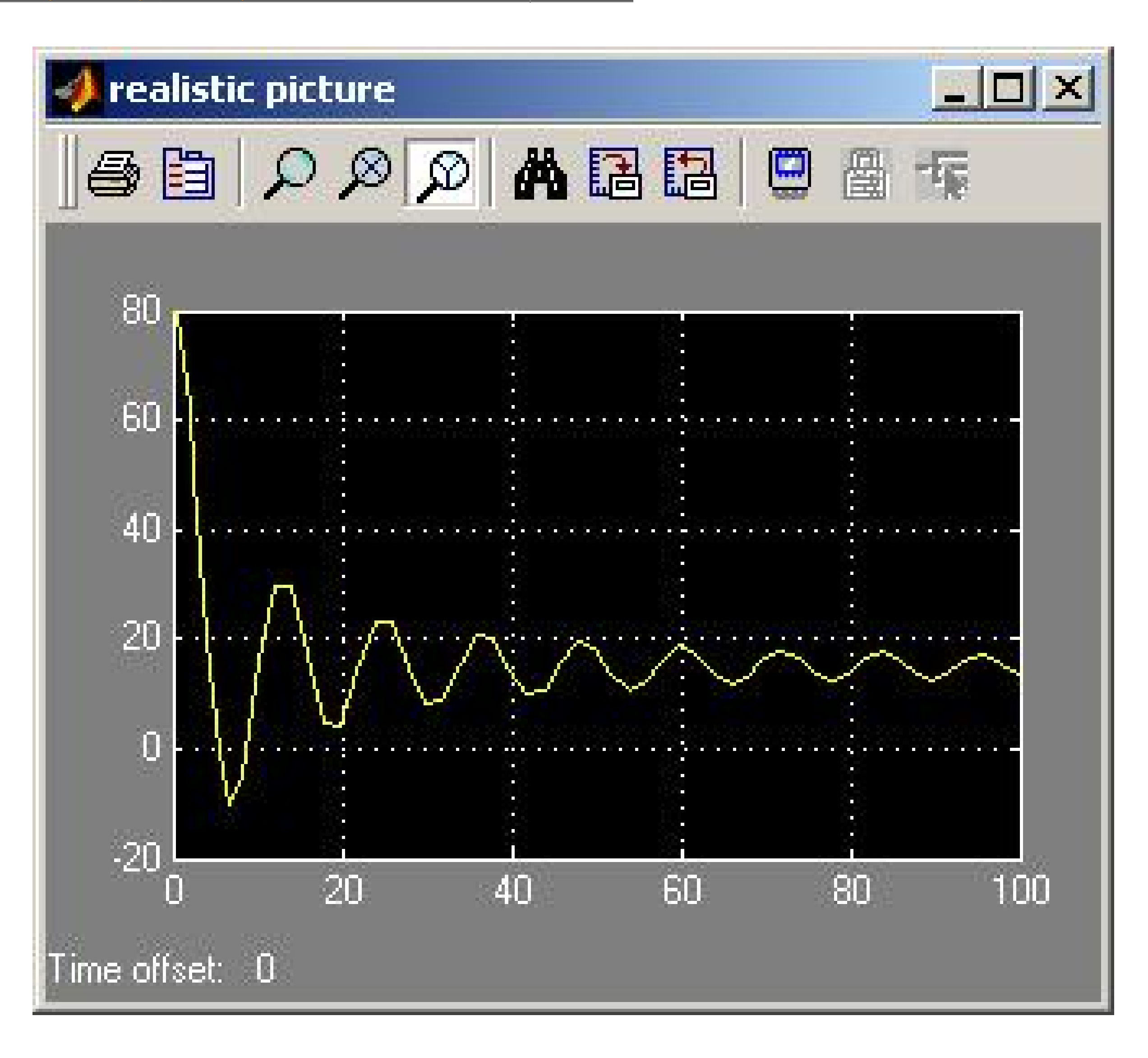


图5.29 蹦极跳系统的仿真结果

5.4.5 仿真精度控制

细心的读者一定会发现，在图5.29蹦极跳系统的仿真结果中，仿真曲线的波峰与波谷处曲线很不光滑。而从蹦极跳系统的数学方程中分析可知，系统的输出曲线应该是光滑曲线。造成这一结果的主要原因是：对此系统仿真来说，连续求解器的默认积分误差取值偏大。因此，只有设置合适的积分误差限，才能获得更好的仿真结果。

变步长连续求解器可以根据积分误差对仿真步长进行自动调整。因此，用户可以设置合适的积分误差上限，以对系统仿真步长进行控制，从而获得更好的仿真结果。积分误差分为如下的两种：

(1) 绝对误差：积分误差的绝对值。

(2) 相对误差：绝对误差除以状态的值。

在仿真参数设置对话框中，用户可以对积分绝对误差与 相对误差进行合适的设置。这样，在系统仿真中求解器只有满足给定的误差条件时才能够进行下一步计算。一般说来，当状态值较大时，相对误差小于绝对误差，由相对误差控制求解器的运行；而当状态值接近零时，绝对误差小于相对误差，则由绝对误差来控制求解器。

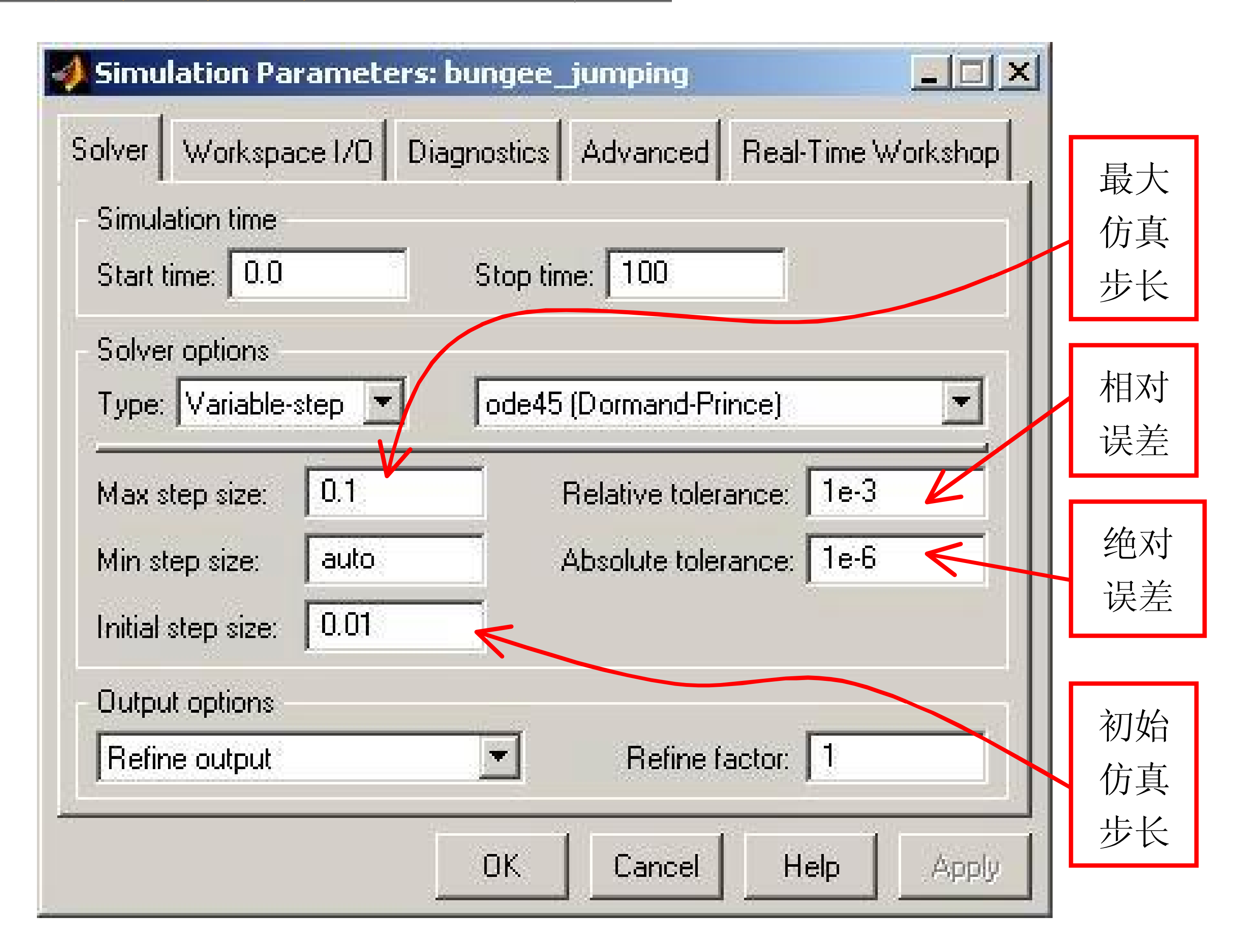


图5.30 蹦极跳系统的仿真参数设置

　　然后再对蹦极跳系统进行仿真，其仿真结果如图5.31所示。从图中可以明显得看出，减小系统仿真积分误差可以有效的提高系统的仿真性能，仿真输出波峰与波谷处的曲线变得比较光滑。

图5.31 新的仿真参数设置下的仿真结果



5.5 线性系统仿真分析

5.1.1线性离散系统仿真分析

随着数字信号处理技术的快速发展与进步，尤其是以数字信号处理芯片为核心的数字系统的设计与使用，使得数字信号处理技术得到了广泛的应用。数字信号处理技术具有诸多模拟信号处理技术所不具备的优点，因此在很多领域都开始取代传统的模拟信号处理。下面以数字滤波器系统为例来介绍线性离散系统的仿真技术。

1. 数字滤波器的数学描述

数字滤波器可以对系统输入的信号进行数字滤波。这里以低通数字滤波器为例说明线性离散系统的仿真技术。低通滤波器可以滤除信号中的高频部分，以获取信号中有用的低频信号，其使用用非常广泛。下面给出一个低通数字滤波器的差分方程描述：

$$y(n)-1.6y(n-1)+0.7y(n-2)=0.04u(n)+0.08u(n-1)+0.04u(n-2)$$

其中 $u(n)$为滤波器的输入， $y(n)$为滤波器的输出。由线性系统的定义可知，此低通数字滤器为一线性离散系统。线性离散系统往往在Z域进行描述，由滤波器系统的差分方程可获得系统的Z变换域描述：

$$\frac{Y(z)}{U(z)}=\frac{0.04+0.08z^{-1}+0.04z^{-2}}{1-1.6z^{-1}+0.7z^{-2}}$$

1. 建立数字滤波器系统模型

这里使用简单的通信系统说明低通数字滤波器的功能。在此系统中，发送方首先使用高频正弦波对一低频锯齿波进行幅度调制，然后在无损信道中传递此幅度调制信号；接收方在接受到幅度调制信号后，首先对其进行解调，然后使用低通数字滤波器对解调后的信号进行滤波以获得低频锯齿波信号。

建立此系统模型所需要的系统模块主要有：

（1）Sources模块库中的Sine Wave模块：用来产生高频载波信号Carrier与解调信号Carrier1。

（2）Sources模块库中的Signal Generator模块：用来产生低频锯齿波信号sawtooth。

（3）Discrete模块库中的Discrete Filter模块：用来表示数字滤波器。

（4）Math模块库中的Product模块：用来完成低频信号的调制与解调。

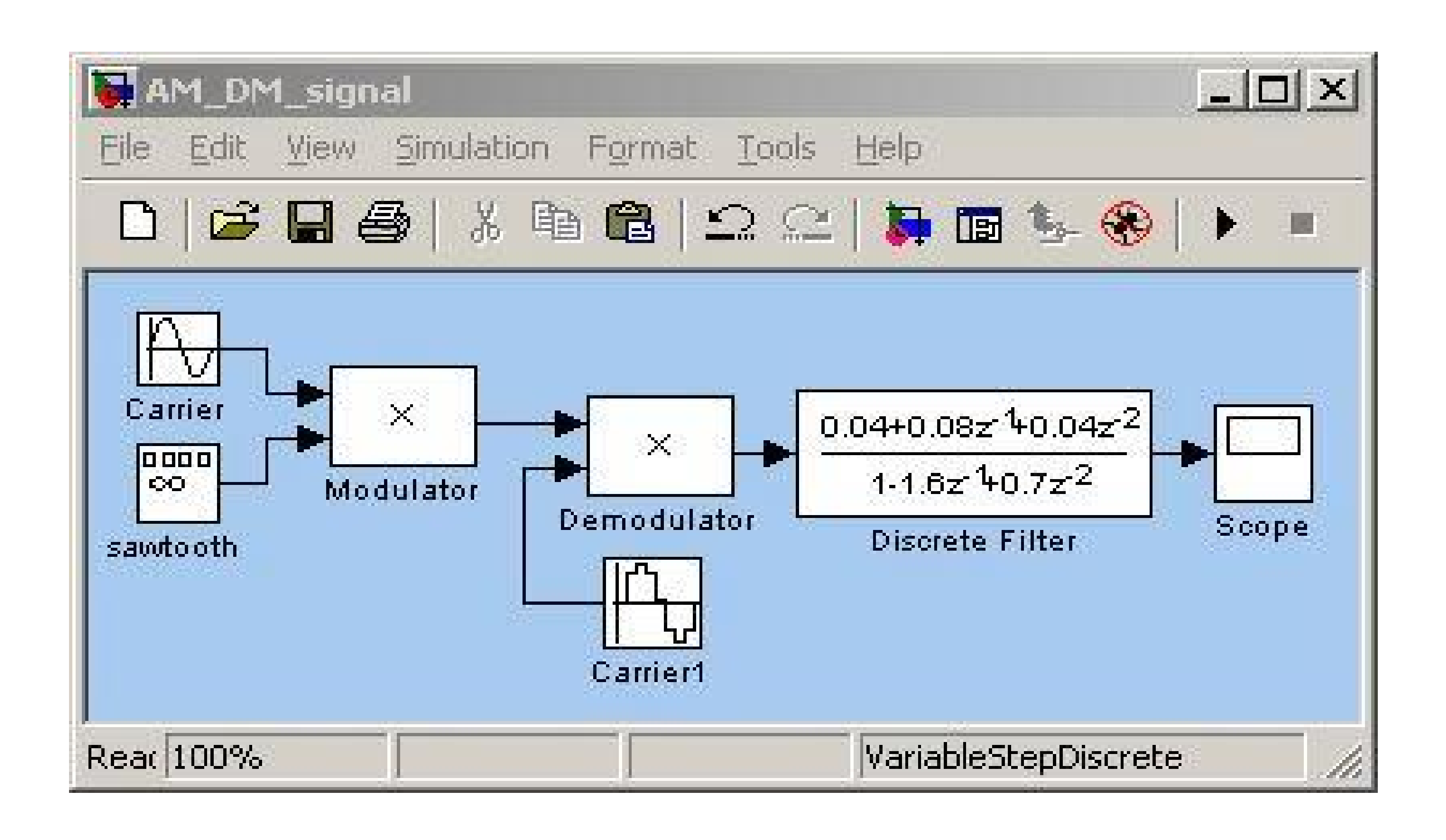


图5.32 数字滤波器系统模型

1．系统模块参数设置

在数字滤波器系统模型建立之后，需要对模型中各个系统模块进行如下的参数设置：

（1）正弦载波信号模块Carrier的参数设置：频率Frequency为1000rad/sec，其余设置为默认值。

（2）信号发生器模块Signal Generator参数设置：Wave form设置为sawtooth，幅值与频率均设置为1（默认值）。

（3）正弦解调信号模块Carrier1参数设置：频率为1000rad/sec，采样时间Sample time为0.005s，其余设置为默认值。

（4）数字滤波器模块Discrete Filter参数数字：分子多项式numerator为[0.04 0.08 0.04]、分母多项式为[1 -1.6 0.7]、采样时间Sample time为0.005s。

由于对模块的参数设置非常的简单，因此这里不再给出相应的模块参数设置对话框；但是需要说明以下几点：

（1）信号发生器模块Signal Generator可以用来产生多种信号如方波信号、正弦信号、锯齿波信号及随机信号等，使用时只需选择相应的信号即可。

（2）解调信号为离散信号，主要是为了使数字滤波器的输入信号为一数字信号。

（3）数字滤波器的采样时间一般应与解调信号的采样时间保持一致。

4．系统仿真参数设置与仿真分析

在系统模块参数设置完毕之后，接下来设置系统仿真参数，最后进行系统的仿真分析。在此使用变步长连续求解器对此系统进行仿真分析。仿真参数设置如图5.33所示。

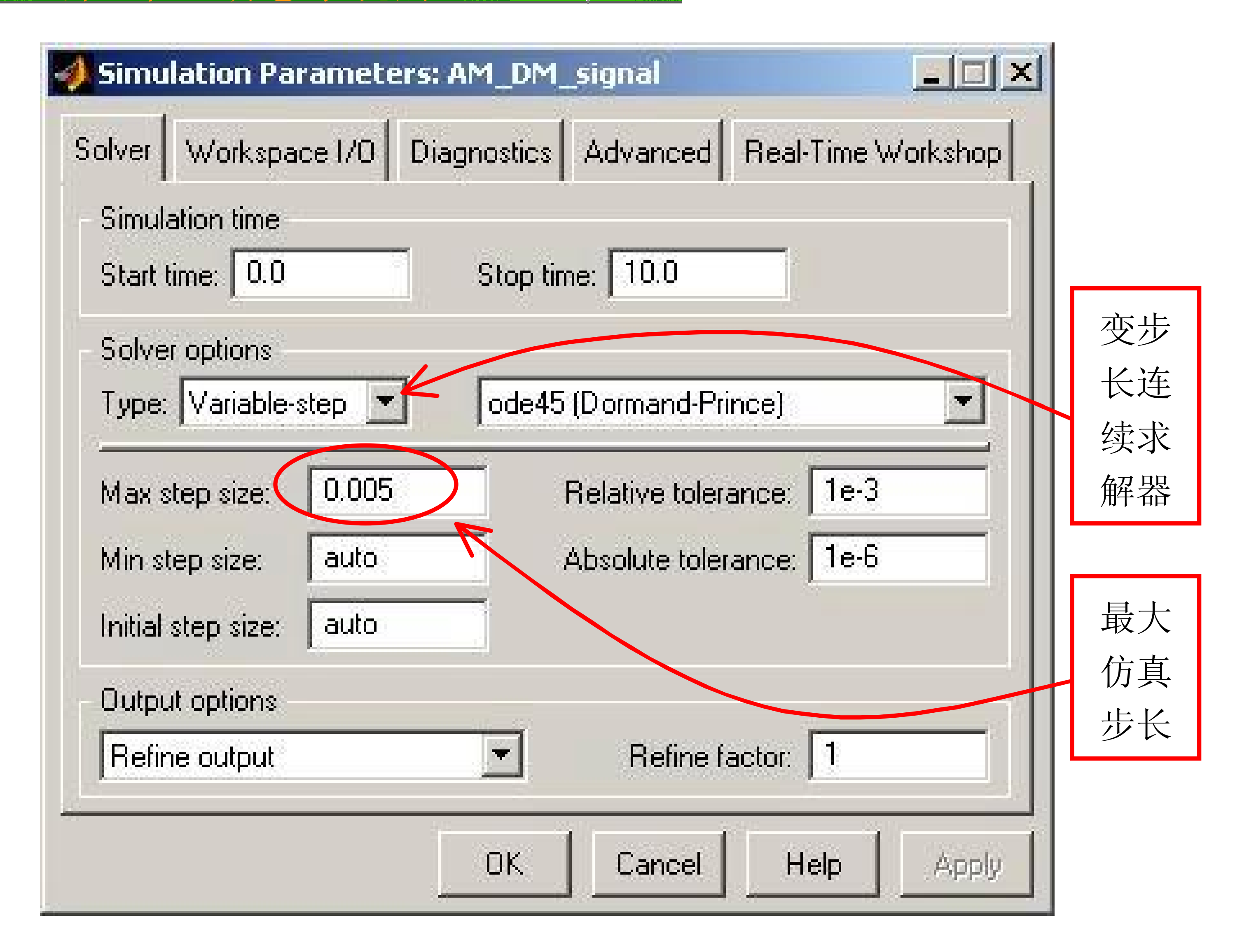


图5.33 数字滤波器系统仿真参数设置

接下来对系统进行仿真，以分析数字滤波器的性能。将原始锯齿波信号与数字滤波器的输出信号（仿真结果）进行比较，如图5.34所示。

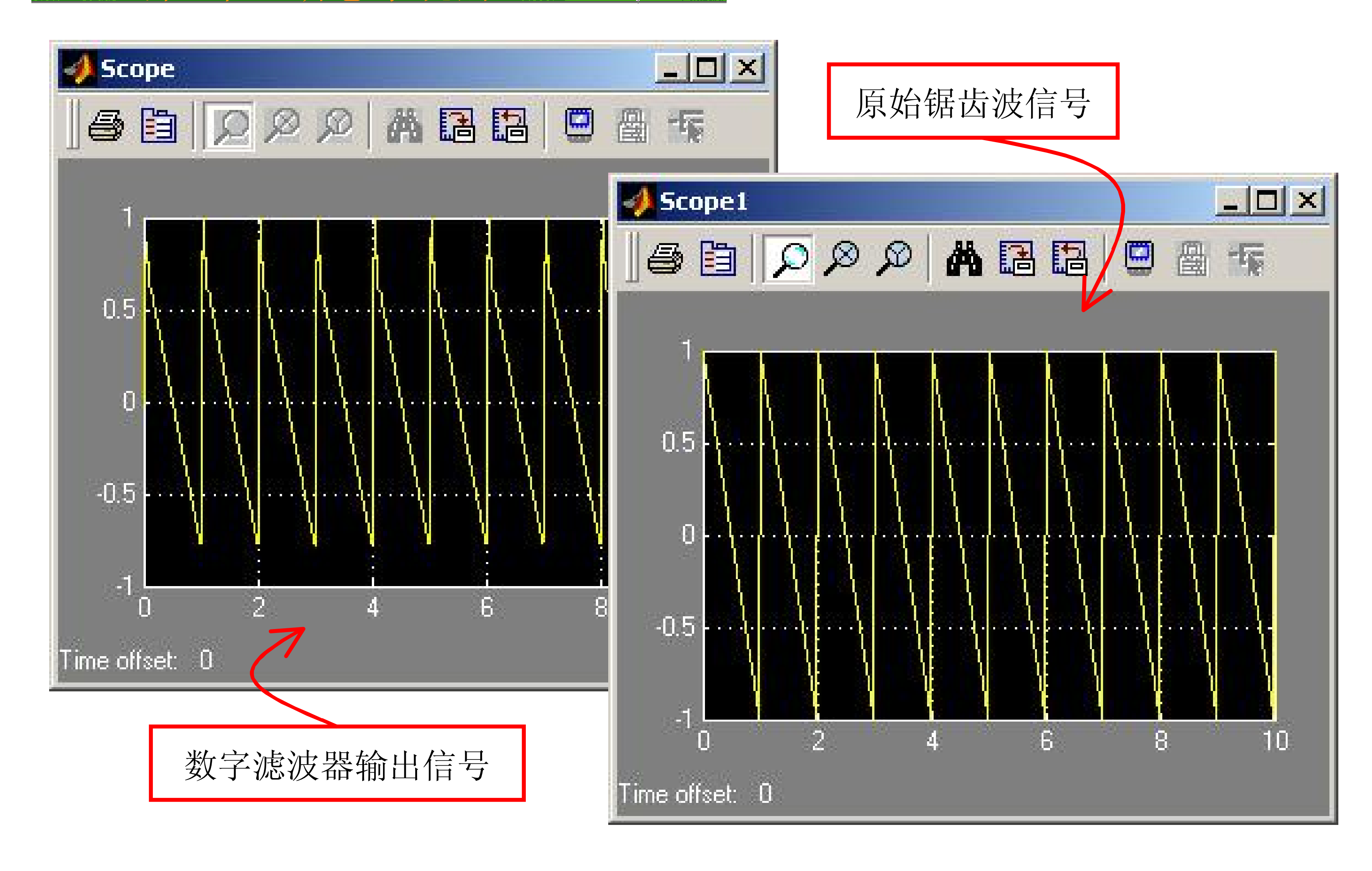


图5.34 系统仿真结果与原始锯齿波信号

am_signal

de_signal

离散单边解调信号

连续双边调制信号

图5.35 连续双边调制信号与离散单边解调信号

5.5.2 线性连续系统仿真分析

第四节中对连续系统的仿真技术做了详细的介绍，对于具有连续状态的连续系统而言，往往使用积分器模块Integrator建立系统的模型。虽然实际的动态系统多数都是非线性系统，但是在一定的范围之内，非线性系统可以由线性系统来近似。由于非线性系统的设计、仿真与分析技术都比较复杂，而且还不成熟，尚无通用的理论形成；而线性系统的设计、仿真分析比较简单，故其应用非常广泛。本部分将介绍线性连续系统的仿真技术。

1. 建立线性连续系统的模型

建立线性连续系统模型的方法与建立一般线性系统模型的方法没有什么本质的区别。不同之处在于，线性连续系统往往使用传递函数模型、零极点模型以及状态空间模型进行描述，因此在建立线性连续系统模型时经常使用与此相应的模块（即Continuous模块库中的Transfer Fcn模块、Zero-Pole模块以及State-Space模块）。下面以比例-微分控制器系统模型的建立为例进行说明。

执行机构

比例微分控制器

图5.36 比例-微分控制系统模型

2．设置系统模块参数与仿真参数

在建立比例微分控制系统模型之后，需要设置各模块参数与系统仿真参数。系统模型中模块参数设置如下：

（1）Zero-Pole模块设置：设置零点Zeros为[ ]、设置极点Poles为[0 -0.5]、设置增益为1。

（2）Step信号模块设置：使用系统的默认取值，即单位阶跃信号。

（3）其它各模块的参数设置如图5.36中所示。

在设置系统模块参数之后，接下来使用Simulation Parameters仿真参数对话框中的Solver选项卡设置系统仿真参数，如下所述：

（1）仿真时间范围为0至20s。

（2）使用变步长连续求解器（variable-step），仿真算法为ode45。

（3）最大仿真步长（Max step size）为0.01。

（4）绝对误差（Absolute tolerance）为1e-6。

（5）其余仿真参数使用默认取值。

1．仿真分析

在对模块参数与系统仿真参数进行合适的设置之后，运行系统仿真，结果如图5.37所示。从系统的仿真结果可以明显的看出，比例-微分控制系统为典型的二阶线性系统；在阶跃信号的作用下，系统不断地对位置误差进行控制修正，最终使系统达到稳定的状态。为改善此系统的性能（如降低系统的超调量、减小系统的过渡时间等），可以对比例-微分控制器进行调整（改变Gain、Gain1模块的增益）以获得更好的性能。

图5.37 比例-微分系统仿真结果



5.6　混合系统设计分析

5.6 .1　混合系统仿真技术的一般知识

在对混合系统进行仿真分析时，必须考虑系统中连续信号与离散信号采样时间之间的匹配问题。Simulink中的变步长连续求解器充分考虑到了连续信号与离散信号采样时间的匹配问题。因此在对混合系统进行仿真分析时，应该使用变步长连续求解器。

由于混合系统中信号类型不一，使用同样的样式表示信号不利于用户对系统模型的理解。Simulink仿真环境提供的Sample time colors功能可以很好的将不同类型、不同采样时间的信号用不同的颜色表示出来，从而可以使用户对混合系统中的信号有个清晰的了解。只需选择Format菜单中的Sample time colors命令便可实现这一功能。其中黑色信号线表示连续信号，其它颜色的信号表示离散信号；并且不同的颜色表示采样时间的不同，其中红色的信号表示采样速率时间最快的，绿色次之，而黄色表示含多速率的系统或是信号。下面举例说明变步长连续求解器对信号采样时间的匹配问题，如图5.38所示。

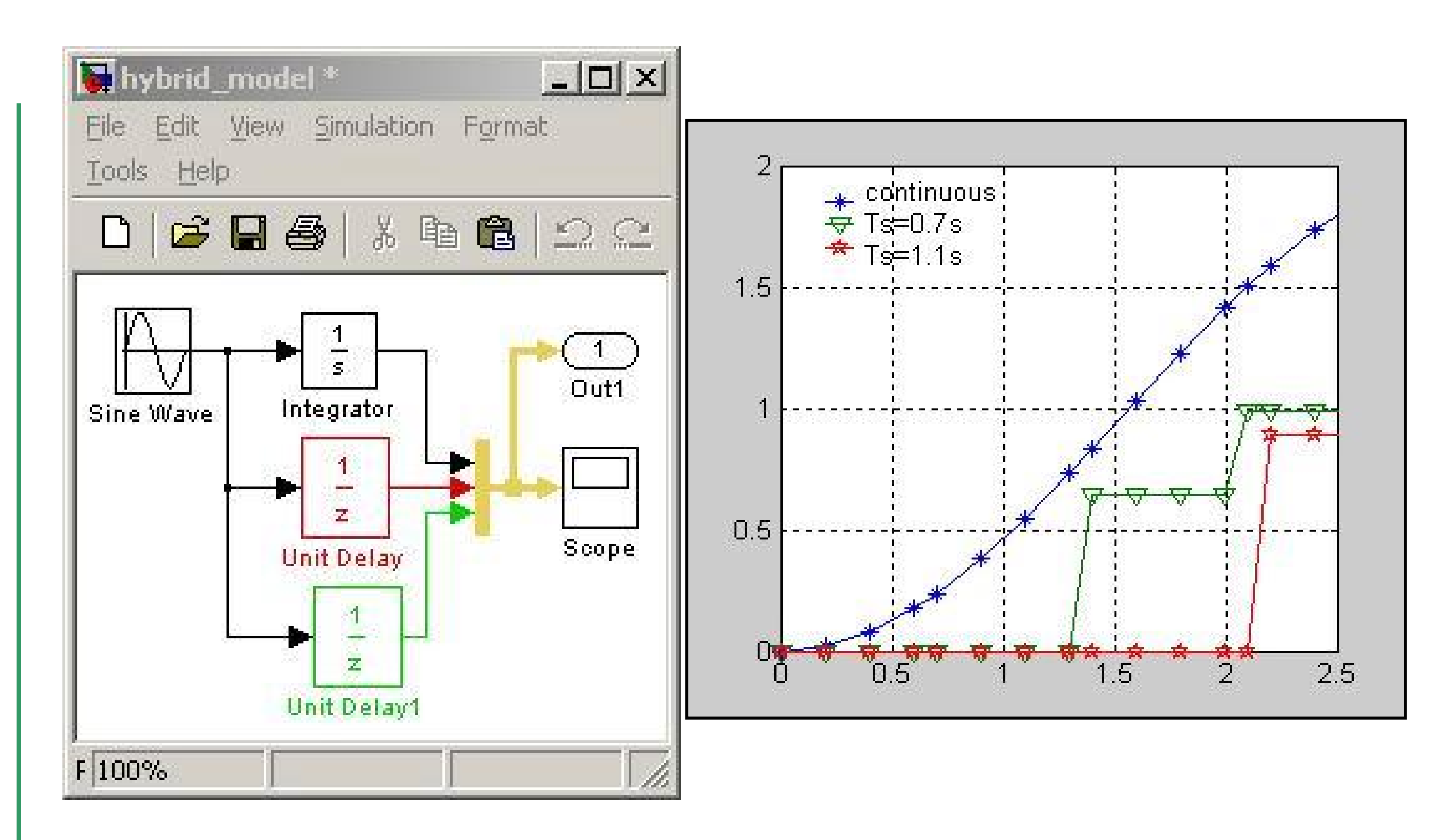


图5.38　变步长连续求解器对信号采样时间的匹配

图中左侧所示为一简单的混合系统模型（使用了Sample time colors功能表示不同采样时间的信号，其中Unit Delay模块采样时间为0.7s，Unit Delay1模块采样时间为1.1s），图中右侧所示为系统仿真输出数据点的图形，从中可以明显的看出：变步长连续求解器不时的减小仿真步长以匹配离散信号的采样时刻。

5.6.2 混合系统设计之一：通信系统

前面以低通数字滤波器为例介绍线性离散系统仿真技术时，我们以无损信道通信系统为例说明了数字滤波器的功能。其实在实际的通信系统中，所有的信道都存在着不同程度的信道噪音，均使信道所传递的信号受到一定的损失。下面对一个实际的通信系统进行仿真分析，以说明混合系统的设计、仿真与分析。

1. 通信系统的物理模型与数学描述

·通信系统的信源

·通信系统的调制与解调

·通信信道

（1）通信信道动态方程为。其中为信道输入，为信道输出。显然，此信道为一线性连续信道，信道传递函数描述如下：

$$\frac{Y(s)}{U(s)}=\frac{1}{10^{-9}s^{2}+10^{-3}s+1}$$

（2）信道噪音：信道受到服从高斯正态分布的随机加性噪音的干扰，噪音均值为0，方差为0.01。

（3）信道延迟：信道经过缓冲区为1024的延迟。

·数字滤波器

数字滤波器的差分方程为

$$y(n)-1.6y(n-1)+0.7y(n-2)=0.04u(n)+0.08u(n-1)+0.04u(n-2)$$

此数字滤波器为线性离散系统，使用滤波器形式对其进行描述如下：

$$\frac{Y(z)}{U(z)}=\frac{0.04+0.08z^{-1}+0.04z^{-2}}{1-1.6z^{-1}+0.7z^{-2}}$$

1. 建立通信系统模型

按照通信系统的物理与数学模型建立系统模型。在建立系统模型之前，首先给出建立系统模型所需要的系统模块，如下所述：

（1）Sources模块库中的Sine Wave模块：作为高频载波信号与解调信号。

（2）Sources模块库中的Signal Generator模块：产生低频锯齿波信号。

（3）Math模块库中的Product模块：用于信号进行调制与解调。

（4）Continuous模块库中的Transfer Fcn模块：描述通信信道。

（5）Sources模块库中的Random Number模块：产生信道噪音。

（6）Continuous模块库中的Transport Delay模块：产生信道延迟。

（7）Discrete模块库中的Discrete Filter模块：描述数字滤波器。

（8）Subsystems模块库中的Subsystem模块：封装系统中不同部分。

（9）Sinks模块库的Scope模块：显示输出。

然后建立系统模型，并将信号幅值调制、通信信道、幅值解调封装到单独的子系统之中，如图5.39所示。

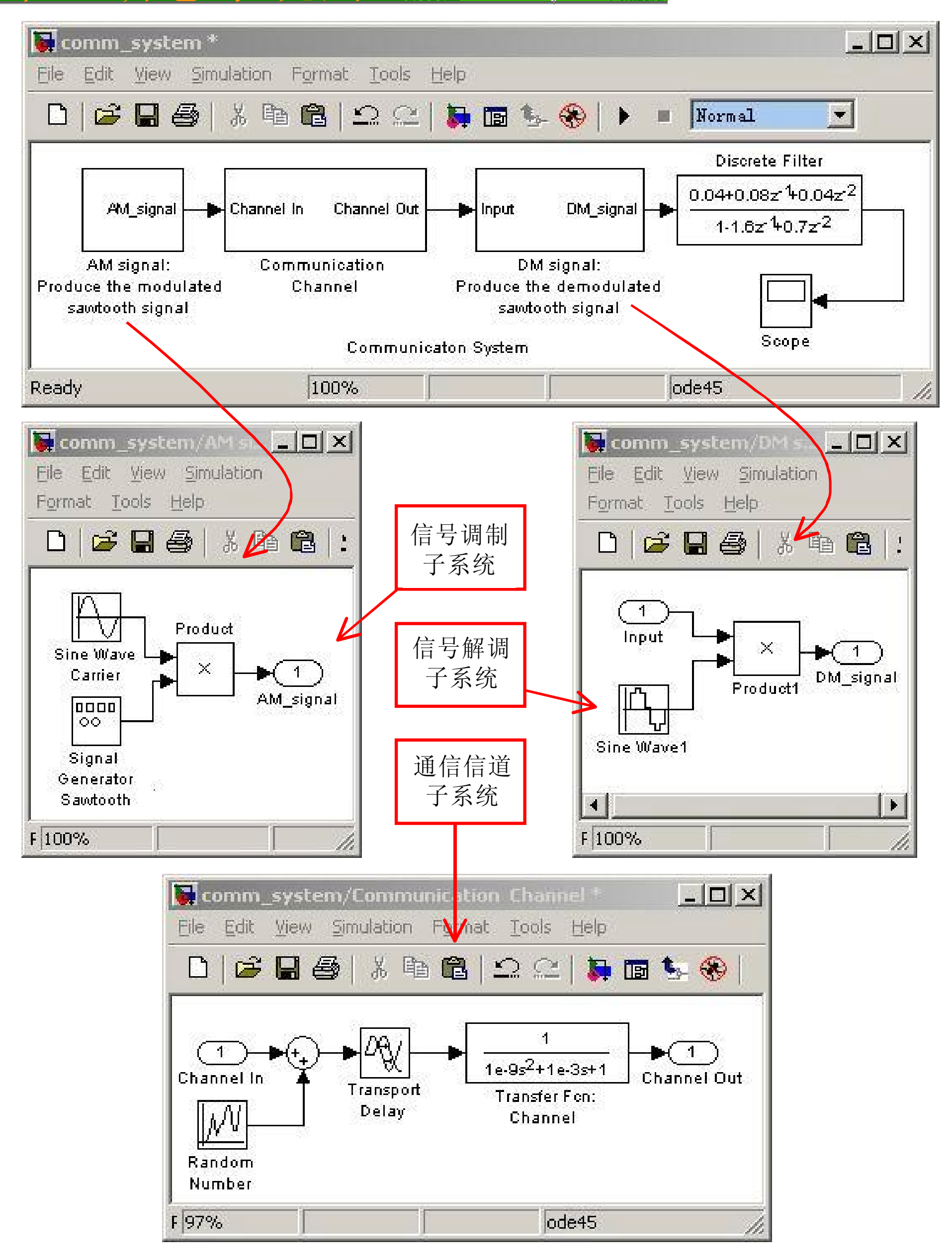

comm_system *
File Edit View Simulation Format Tools Help
Normal
AM_signal
Channel In
Channel Out
Input
DM_signal
Discrete Filter
0.04+0.08z-1+0.04z-2
1-1.6z-1+0.7z-2
AM signal:
Produce the modulated
sawtooth signal
Communication
Channel
DM signal:
Produce the demodulated
sawtooth signal
Scope
Communicaton System
Ready
100%
ode45
信号调制
子系统
信号解调
子系统
通信信道
子系统
Sine Wave
Carrier
Product
AM_signal
Signal
Generator
Sawtooth
Input
Product1
DM_signal
Sine Wave1
comm_system/Communication Channel *
Channel In
Random
Number
Transport
Delay
1
1e-9s2+1e-3s+1
Transfer Fcn:
Channel
Channel Out
97%

图5.39 通信系统模型

3. 系统模块参数设置与仿真参数设置

在建立系统模型之后，接下来按照系统的要求设置系统模块参数与仿真参数。想必用户对模块参数设置与仿真参数设置已经比较熟悉了，故在此仅给出各模块的参数与相应的仿真参数（所有没有给出的模块参数或仿真参数均使用系统的默认值）；用户可仿照前面的例子对参数进行正确的设置。

信号调制子系统参数

（1）正弦载波Sine Wave模块：频率Frequency为100Hz，幅值为1。

（2）锯齿信号Signal Generator模块：波形Wave form为sawtooth（锯齿波）。

·通信信道子系统参数

（1）随机信号Random Number模块：均值mean为0、方差Variance为0.01。

（2）信道延迟Transfer Delay模块：初始缓冲区Initial buffer size为1024。

（3）信道传递函数Transfer Fcn模块：分子Numerator为[1]，分母Denominator为[1e-9 1e-3 1]。

·信号解调子系统参数

正弦解调信号Sine Wave1模块：频率Frequency为100Hz，幅值为1，采样时间Sample time为0.005s。

·数字滤波器参数

数字滤波器Discrete Filter模块参数：分子Numerator为[0.04 0.08 0.04]、分母Denominator为[1 -1.6 0.7]、采样时间Sample time为0.005s

·系统仿真参数

（1）系统仿真时间：从0至10s。

（2）仿真求解器：变步长连续求解器。

（3）绝对误差：1e-6。

（4）最大仿真步长：0.01。

1．系统仿真与分析

在对系统模块参数与系统仿真参数设置之后，接下来对系统进行仿真分析。为了对通信系统的整体性能有一个直观的认识，这里将系统仿真结果（即通信系统的输出信号）与原始的锯齿波信号（通信系统所要传递的信号）进行比较，如图5.40所示。从图中可以看出，由于通信信道的延迟以及加性随机噪音的干扰，使得通信系统的输出信号比原始锯齿波信号的起始时间慢1s，而且存在一定的失真；但只要失真小于一定的阈值，不会对锯齿波信号的使用造成太大的影响。

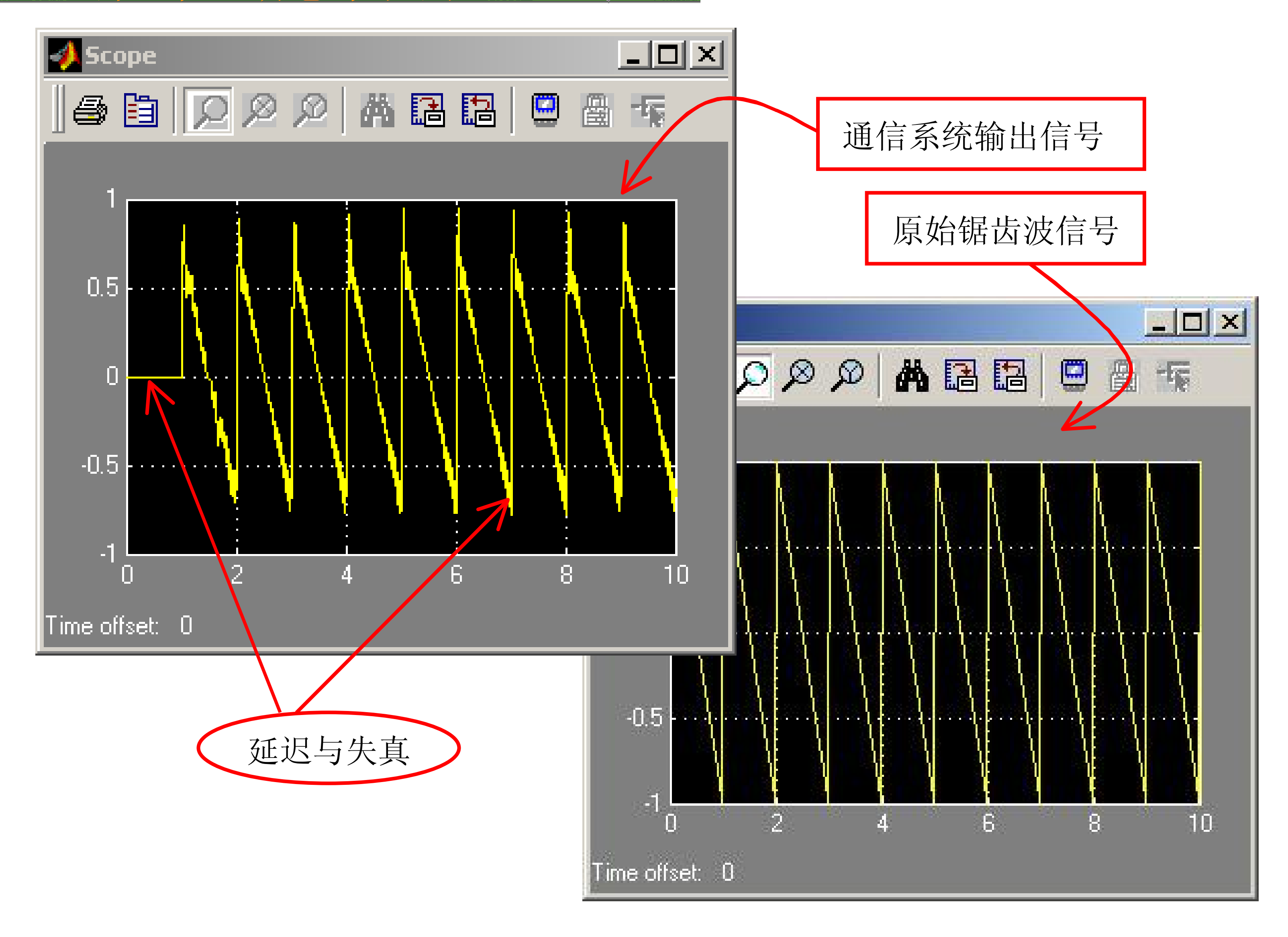


图5.40 通信系统输出与原始锯齿波信号比较

5.6.3 混合系统设计之二：行驶控制系统

汽车行驶控制系统是应用非常广泛的控制系统之一，其主要目的是对汽车速度进行合理的控制。系统的工作原理如下：

（1）汽车速度操纵机构的位置发生改变以设置汽车的速度，这是因为操纵机构的不同位置对应着不同的速度。

（2）测量汽车的当前速度，并求取它与指定速度的差值。

（3）由速度差值信号驱动汽车产生相应的牵引力，并由此牵引力改变汽车的速度直到其速度稳定在指定的速度为止。

由系统的工作原理来看，汽车行驶控制系统为典型的反馈控制系统。下面建立此系统的Simulink模型并进行仿真分析。

1．汽车行驶控制系统的物理模型与数学描述

·速度操纵机构的位置变换器

位置变换器是汽车行驶控制系统的输入部分，其目的是将速度操纵机构的位置转换为相应的速度，二者之间的数学关系如下所示：

$$v = 50x + 45 \qquad x \in [0,1]$$

·离散行驶控制器

行驶控制器是整个汽车行驶控制系统的核心部分。简单说来，其功能是根据汽车当前速度与指定速度的差值，产生相应的牵引力。行驶控制器为一典型的PID控制器，其数学描述为：

积分环节： $x(n) = x(n-1) + u(n)$

微分环节： $d(n) = u(n) - u(n-1)$

系统输出： $y(n) = Pu(n) + Ix(n) + Dd(n)$

·汽车动力机构

汽车动力机构是行驶控制系统的执行机构。其功能是在牵引力的作用下改变汽车速度，使其达到指定的速度。牵引力与速度之间的关系为：

$$F = m\dot{v} + bv$$

其中为汽车的速度、为汽车的牵引力、为汽车的质量、为阻力因子。

1．建立汽车行驶控制系统的模型

按照汽车行驶控制系统的物理模型与数学描述建立系统模型。在进行系统模型之前，首先给出建立系统模型所需的主要系统模块：

（1）Math模块库中的Slider Gain滑动增益模块：对位置变换器的输入信号的范围进行限制。

（2）Discrete模块库中的Unit Delay单位延迟模块：用来实现行驶控制器（即PID控制器）。

（3）Continuous模块中的Integrator积分器模块：用来实现汽车动力机构。

（4）Subsystems模块库中的Subsystem子系统模块：用来对系统不同的部分进行封装。

然后建立系统模型，并将速度操纵机构的位置变换器、行驶控制器、汽车动力机构封装到不同的子系统之中，如图5.41所示。

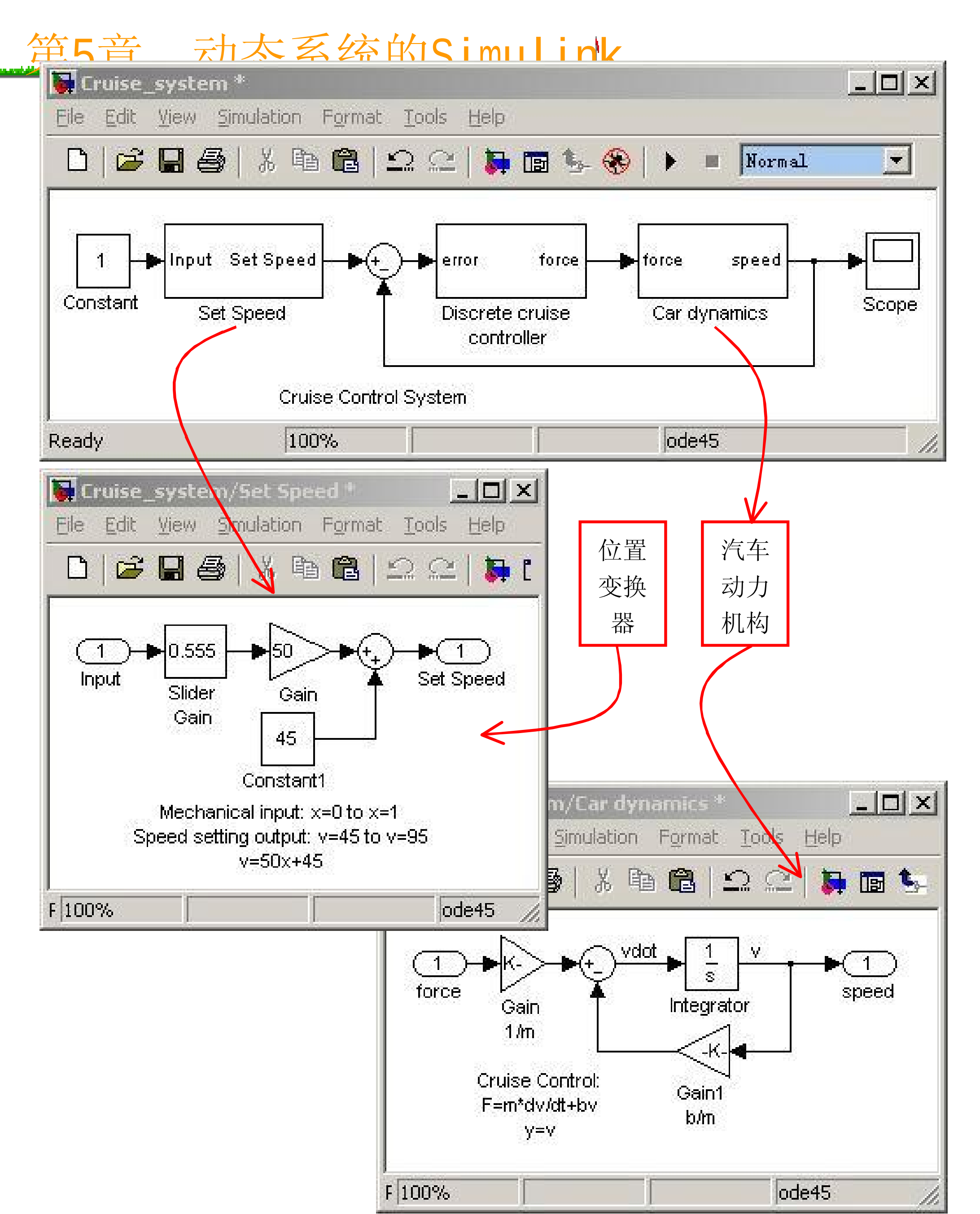


图5.41A 汽车行驶控制系统之位置变换器与汽车动力机构

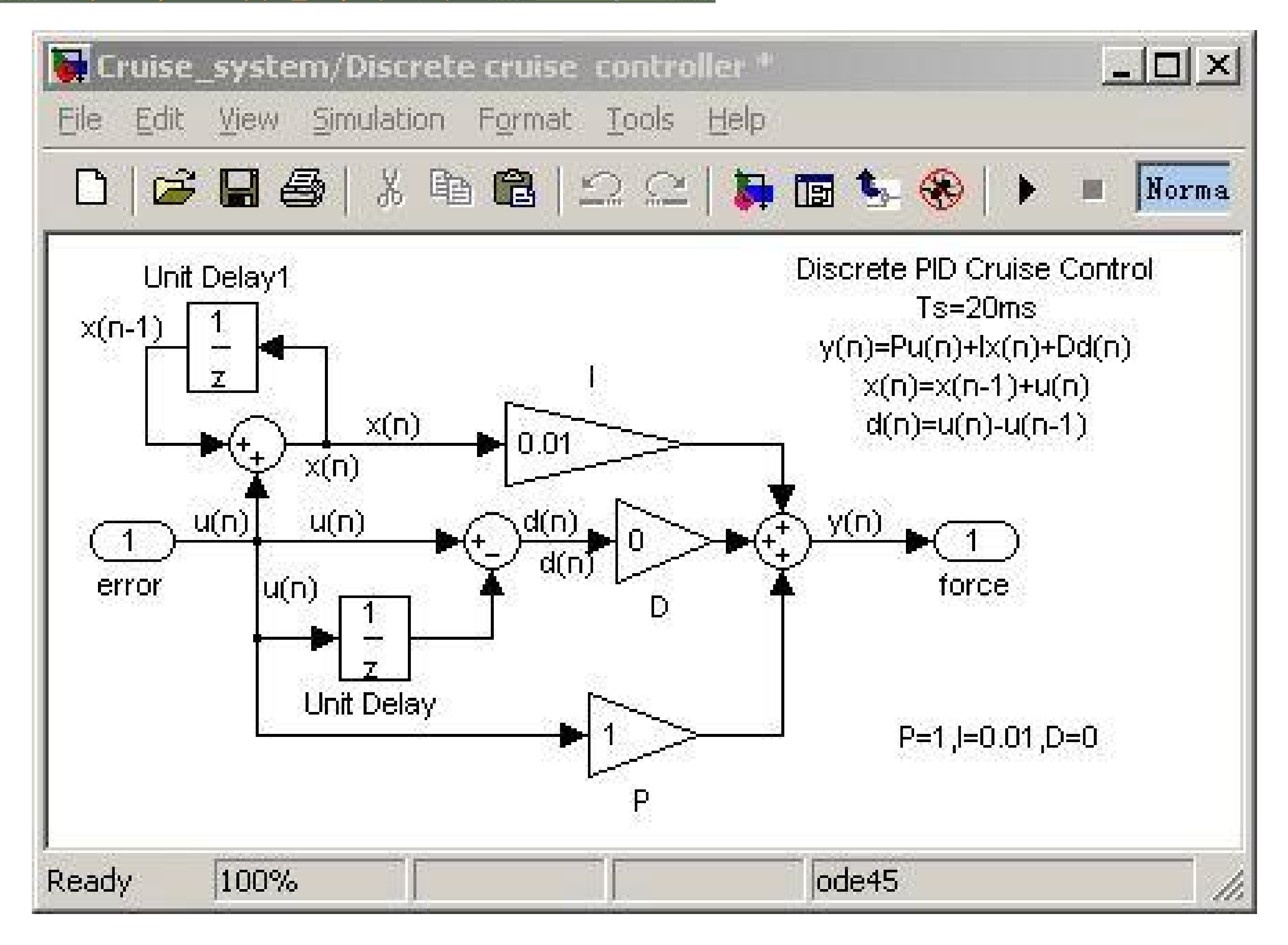


图5.41 B 行驶控制系统之行驶控制器

3.　系统模块参数设置与仿真参数设置

在建立系统模型之后，接下来按照系统的要求设置系统模块参数与仿真参数。这里仅给出模块参数与相应的仿真参数。

·速度操纵机构的位置变换器参数

（1）Slider Gain模块：最小值Low为0、最大值High为1、初始取值0.555。

（2）Gain模块：增益取值为50。

（3）Constant1模块：常数取值为45。

·行驶控制器参数

（1）所有Unit Delay模块：初始状态为0、采样时间为0.02s。

（2）P、I、D增益模块：取值分别为1、0.01、0。

·汽车动力机构参数

（1）Gain模块：取值为1/m，即1/1000。

（2）Gain1模块：取值为b/m，即20/1000。

（3）Integrator模块：初始状态为0，即速度初始值为0。

·系统仿真参数

（1）仿真时间范围：从0至1000s。

（2）求解器：使用变步长连续求解器。

4．系统仿真与分析

在对系统模块参数与系统仿真参数设置之后，接下来对系统进行仿真分析。为了使用户对离散行驶控制器的作用有一个直观的认识，这里使用两组不同的PID控制参数对系统进行仿真，其结果如图5.42所示。

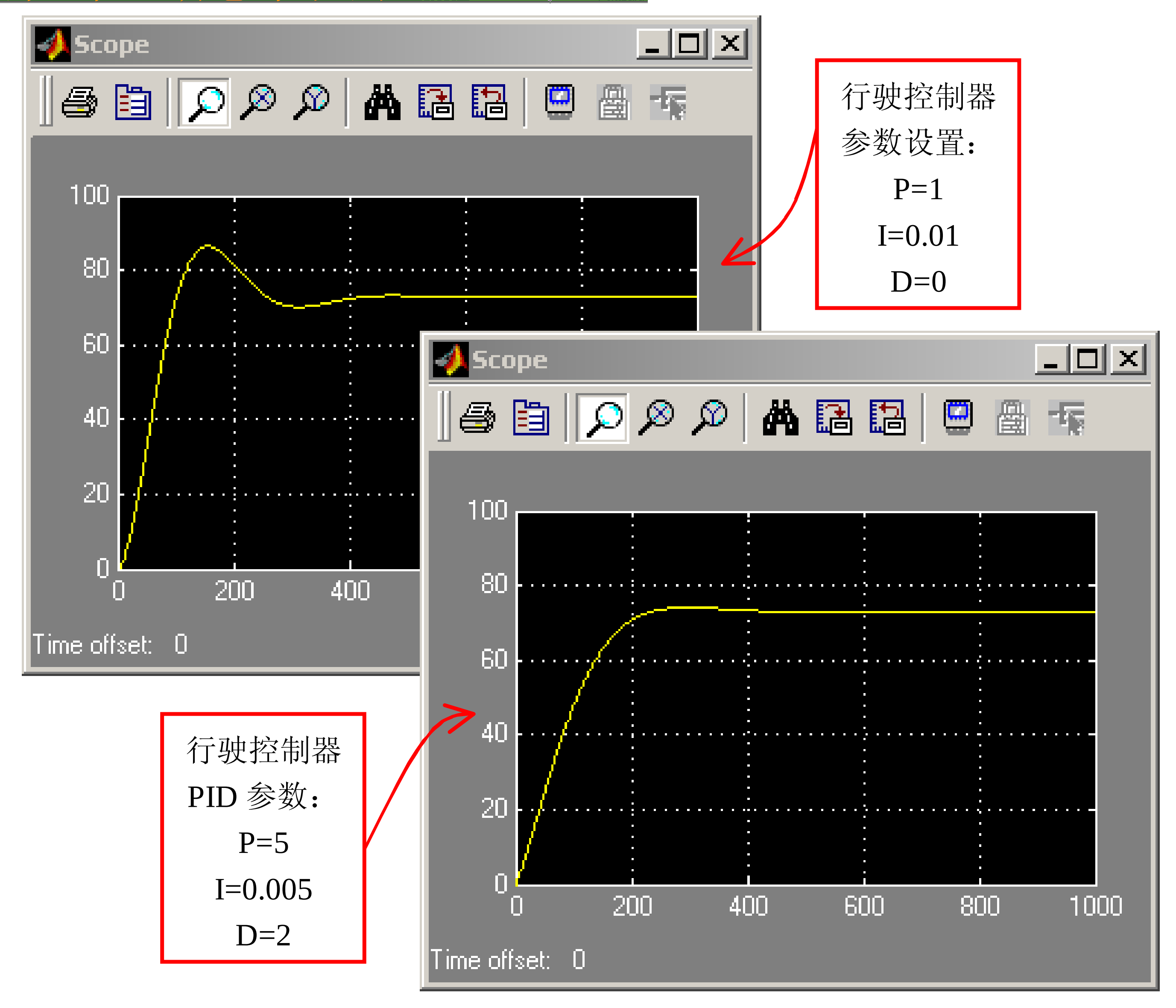


图5.42 不同控制参数下的仿真结果

汽车行驶控制系统的目的是使汽车的速度在较短的时间内平稳地达到指定的速度。从图5.42的仿真结果中可以看出，在行驶控制器控制参数取值为P=1、I=0.01、D=0时，汽车的速度并非直接达到指定的速度，而是经过一个振荡衰减过程，最后逐渐过渡到指定的速度。此时行驶控制系统为典型的二阶欠阻尼控制系统。

5.6.4 工作空间输入输出Workspace I/O设置

在对动态系统进行仿真分析时，往往需要对系统的仿真结果进行进一步的定量分析。使用Scope模块可以直接显示系统仿真结果，非常有利于对系统的定性分析，但并不利于系统的定量分析。本书第4章中介绍的Simulink与Matlab的接口设计技术，允许Simulink与Matlab之间进行数据交互，如从Matlab工作空间中获得系统模块参数、输出仿真结果至Matlab工作空间等。

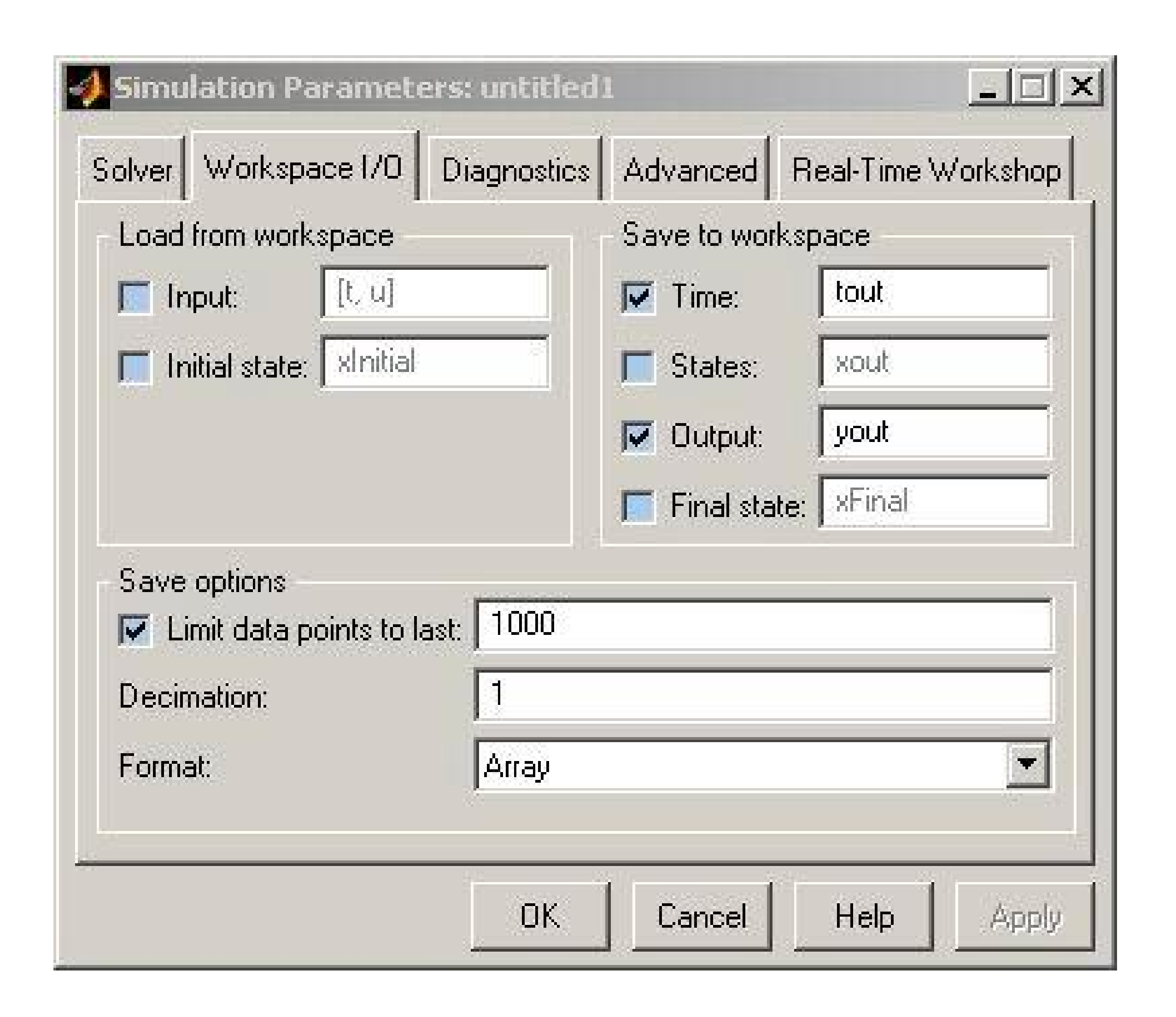


图5.43 Workspace I/O选项卡

1．从Matlab工作空间获得系统输入（Load form workspace）

虽然Simulink提供了多种系统输入信号，但并不能完全满足需要。Simulink允许使用用户自定义的信号作为系统输入信号。在Load form workspace框中，用户可以设置Matlab中的变量作为系统输入信号或是系统状态初始值，如下所述：

（1）Input：用来设置系统输入信号。其格式为[t，u]，其中t、u均为列向量，t为输入信号的时间向量，u为相应时刻的信号取值，可以使用多个信号输入，如[t，u1，u2]。输入信号与Simulink的接口由Inport模块（In1模块）实现。

（2）xInitial　state：用来设置系统状态变量初始值。初始值xInitial可为行向量。

注意，使用xInitial　state所设置状态变量初始值会自动覆盖系统模块中的设置。另外，输入信号与状态变量需要按照系统模型中Inport模块（即In1模块）的顺序进行正确设置。

2．输入数据至Matlab工作空间（Save to workspace）

使用Workspace I/O选项卡可以将系统的仿真结果、系统仿真时刻、系统中的状态或是指定的信号输出到Matlab工作空间之中，以便用户对其进行定量分析，如下所述：

（1）Time：输出系统仿真时刻。

（2）States：输出系统模型中所有的状态变量。

（3）Output：输出系统模型中所有由Outport模块（即Out1模块）表示的信号。

（4）Final state：输出系统模型中的最终状态变量取值，即最后仿真时刻处的状态值。

2．　数据保存设置（Save option）

（1）Limit data points to last：表示输出数据的长度（从信号的最后数据点记起）。

（2）Format：表示输出数据类型。共有三种形式：Structure with Time（带有仿真时间变量的结构体）、Structure（不带仿真时间变量的结构体）以及Array（信号数组）。

上面简单介绍了Workspace I/O选项卡的设置与功能，下面举例说明。对于图5.44所示的系统，按照图中所示设置系统仿真参数中的Workspace I/O选项卡，以从Matlab工作空间获取系统输入信号、系统状态初始值并将系统仿真结果输出至Matlab工作空间之中。

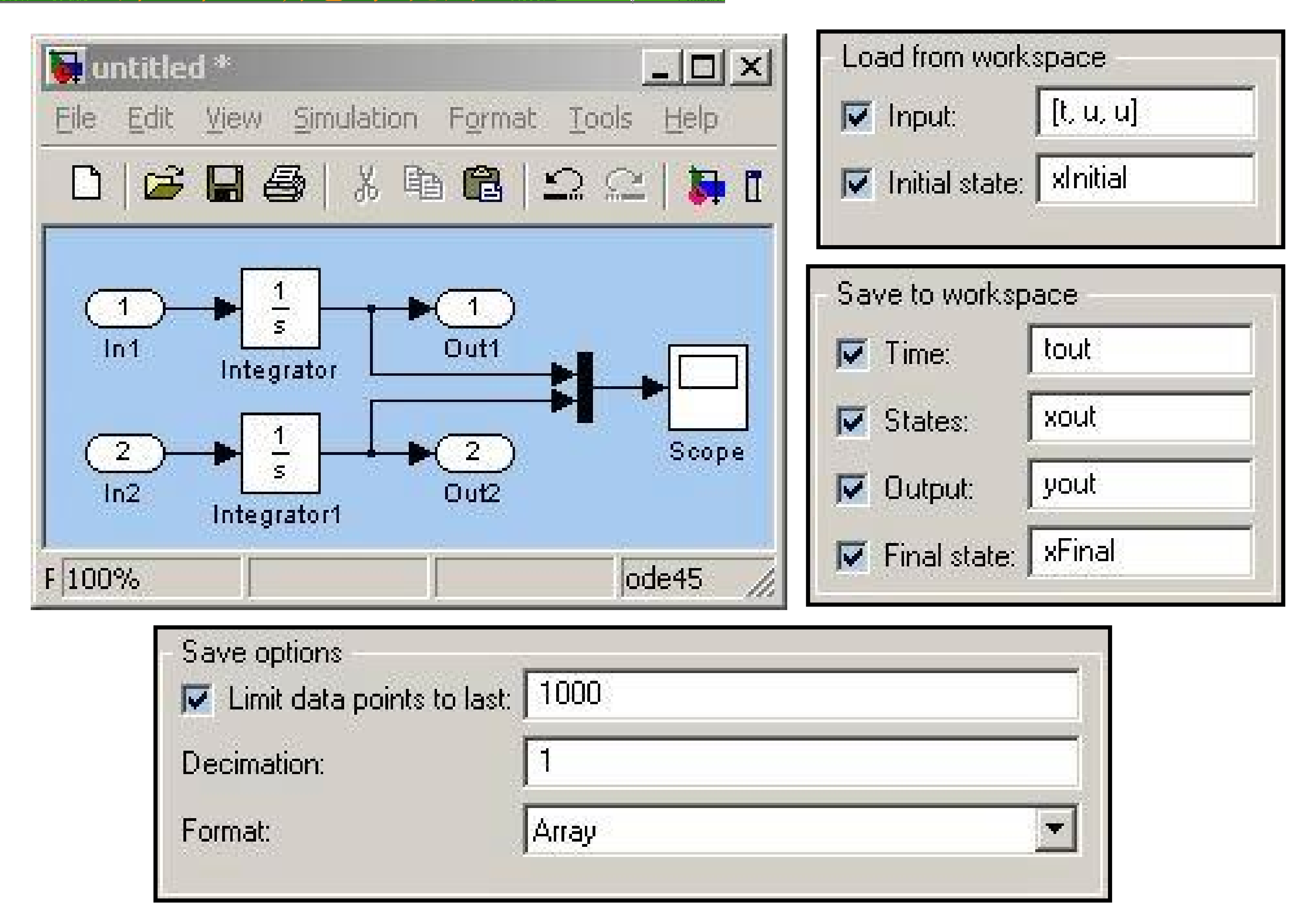


图5.44 Workspace I/O设置举例

在运行仿真之前，首先需要生成系统输入信号与状态初始值，在Matlab命令窗口中键入如下命令：

```
>> t=1:0.1:10; t=t';
>> u=sin(t);  u=u';
>> xInitial=[0,1];
```

然后运行系统仿真，为了观察Workspace I/O设置的效果，这里使用Scope模块显示仿真结果，如图5.45所示。



图5.45 使用Workspace I/O设置的仿真结果

此时查看Matlab的工作空间，用户可以发现系统仿真结果（仿真时间、系统状态、系统运算结果以及最终状态等）被正确的输出到Matlab工作空间中，这里不再赘述。



5.7 Simulink的调试技术

毫无疑问，功能强大、界面友好的调试功能是优秀系统设计开发平台所必备的条件之一。Simulink作为高性能的系统设计、仿真与分析平台，给用户提供了强大的模型调试工具。通过Simulink的调试工具，用户可以对动态系统的系统模型进行调试，以发现其中可能存在的问题，然后进行修改，从而快速完成系统设计、仿真与分析的目的。

不同领域中的不同的系统模型，其复杂程度往往相差悬殊，对系统模型调试的复杂程度也大不相同。Simulink所提供的图形调试器可以满足多数应用领域中系统模型的调试，而并非针对专门的应用领域所设计的。因此，在介绍Simulink调试器功能时，这里仅以最为简单的例子对其进行说明，以便具有不同专业背景的系统设计人员都可以很好的理解。

5.1.1Simulink图形调试器启动Simulink的图形调试器具有优秀的用户界面，使用菜单Tools下的Simulink debugger命令或是使用调试器按钮启动调试器，如图5.46所示。

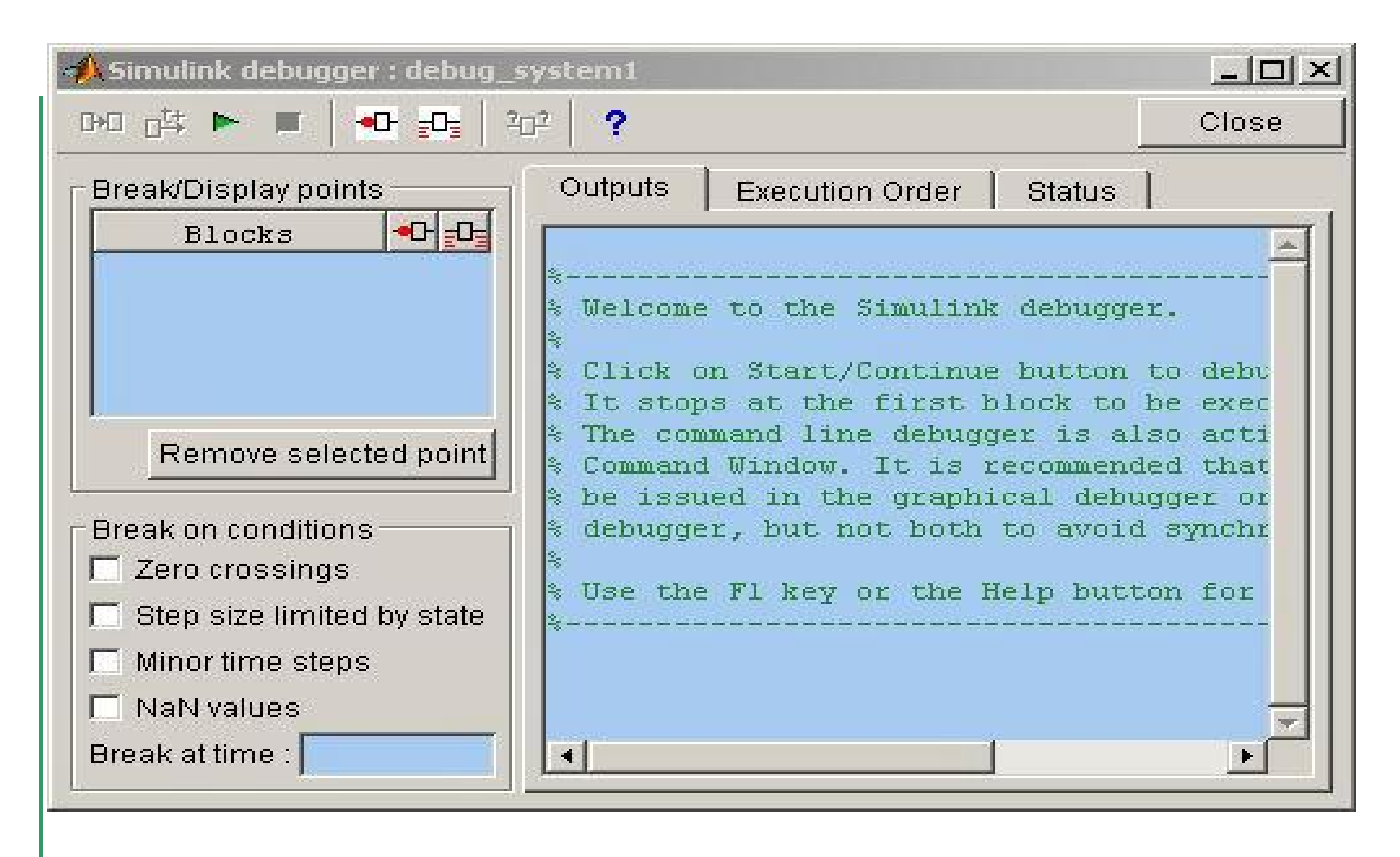


图5.46 Simulink调试器窗口

5.7.2 调试器的操作设置与功能

启动Simulink调试器，设置合适的调试断点之后，便可以对系统模型中指定的模块或信号进行调试了。在设置断点进行调试之前，首先对Simulink图形调试器中的操作设置与功能做一个简单的介绍。

1．　Simulink调试器工具栏

Simulink调试器工具栏命令功能介绍如图5.47所示。

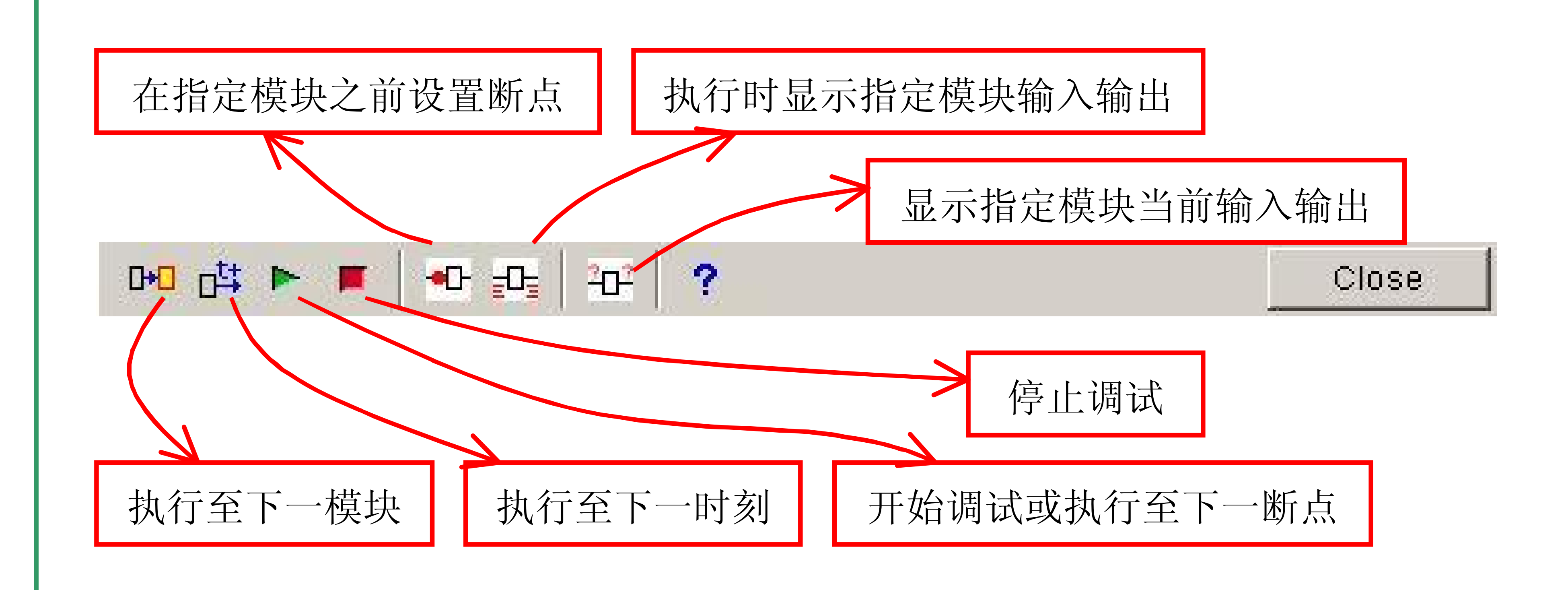


图5.47 Simulink调试器工具栏命令介绍

2.　断点显示及断点条件设置

Simulink提供了友好的调试界面，用户可以在断点显示框中了解到当前断点的信息，如断点位置、断点模块的输入输出等，如图5.48所示。

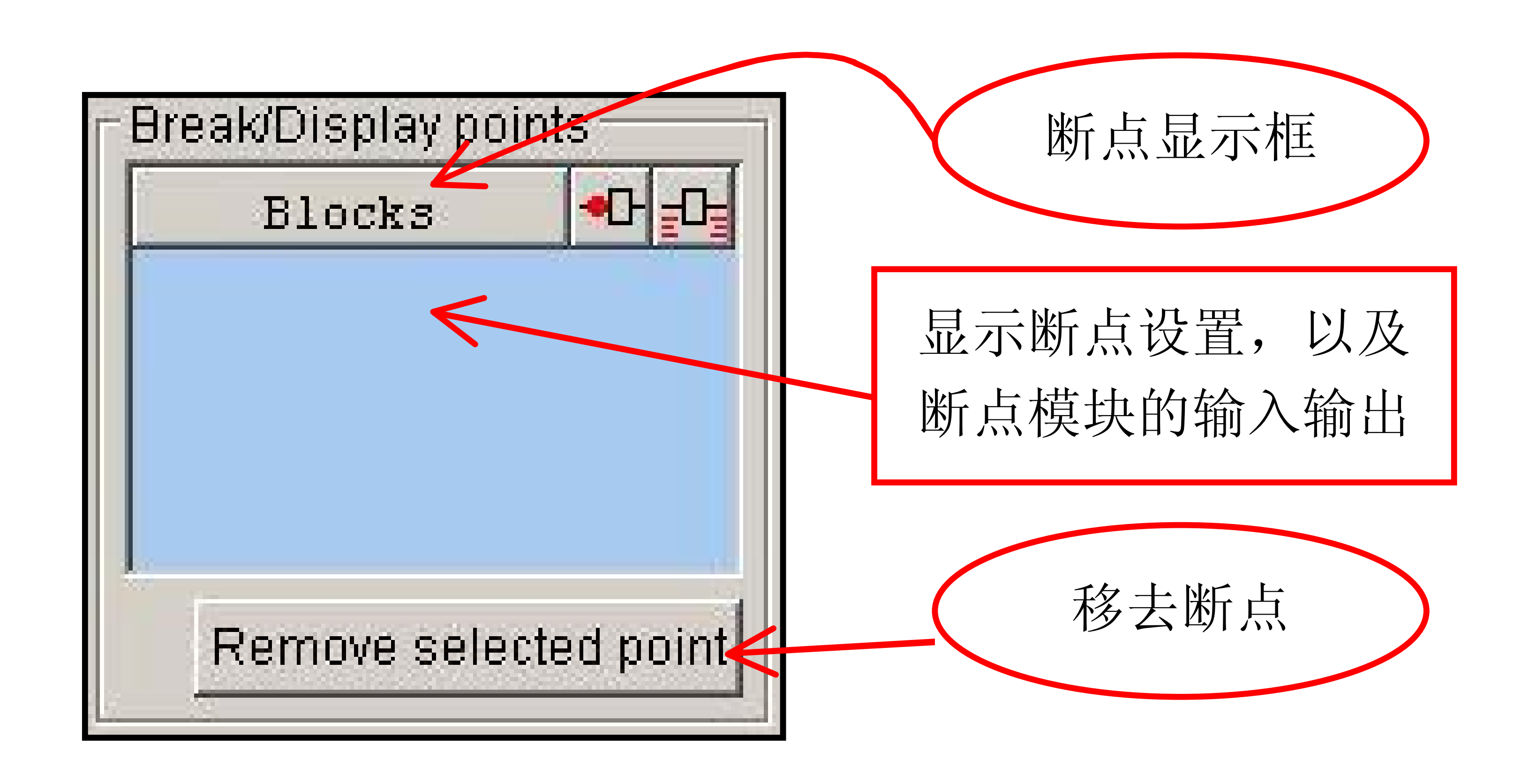

Break/Display points
Blocks
Remove selected point
断点显示框
显示断点设置，以及断点模块的输入输出
移去断点

图5.48 断点显示框

一般说来，用户可以在调试之前在指定的模块之前设置断点。但是多数情况下，用户需要在一定的条件下设置系统断点以进行调试。Simulink调试器提供了五种断点条件设置，如图5.49所示。

Break on conditions
- Zero crossings
- Step size limited by state
- Minor time steps
- NaN values

Break at time :

在系统发生过零处设置断点
在仿真步长受到状态限制处设置断点
在最小仿真步长出现处设置断点
在系统中出现无穷小值处设置断点
在指定的仿真时刻处设置断点

图5.49 断点条件设置

3.　调试器输出窗口

在对指定的系统模型进行调试时，调试结果均在Simulink的输出窗口显示。图5.50所示为Simulink的调试器输出窗口：

调试结果输出

调试执行顺序

调试状态

Outputs | Execution Order | Status

```
%---------------------------------------------------------
% Welcome to the Simulink debugger.
%
% Click on Start/Continue button to debug th
% It stops at the first block to be executed
% The command line debugger is also active i
% Command Window. It is recommended that deb
% be issued in the graphical debugger or the
% debugger, but not both to avoid synchroniz
%
% Use the F1 key or the Help button for add:
%---------------------------------------------------------
```

图5.50　调试器输出窗口

下面对其进行简单的介绍：

（1）Outputs：输出调试结果，如调试时刻、调试的模块以及模块输入输出等。

（2）Execution Order：输出调试顺序，即调试过程中各模块的执行顺序。

（3）Status：输出调试状态，如当前仿真时间、缺省调试命令（执行至下一模块或是执行至下一时间步）、调试断点设置以及断点数等状态信息。

5.7.3 系统调试举例

从对调试器操作设置与功能的介绍中可以看出，使用Simulink图形调试器既简单又直观。接下来以图5.51所示的系统模型为例说明Simulink的调试技术，系统模型如图5.51A和子系统5.51B所示。

图5.51 A 系统模型

图5.51 B 系统模型的子系统实现

1．设置系统调试断点

（1）启动Simulink的调试器。

（2）在子系统中增益模块（Gain模块）之前设置断点：选择Gain模块，然后单击Simulink调试器工具栏中‘在指定模块之前设置断点’图标即可。此时，断点显示框中将显示断点设置情况，如图5.52所示。

图5.52 系统调试断点设置

3.　逐模块执行调试

系统模型一旦进入调试模式，使用调试器工具栏中的第一个按钮‘执行下一模块’，便可逐模块对系统进行调试。在调试过程中，即将被执行的模块系统会用黄颜色突出显示。当此模块被执行完毕时，在调试器输出窗口中将显示相应的系统仿真时刻、模块的输入与输出。如果系统调试至系统模型中的子系统，则调试器会自动打开相应的子系统。图5.53所示为逐模块执行系统调试的示意图。

仿真时刻 Tm 为 0,调试模块 Sine Wave

Sine Wave 模块执行完毕输出结果

仿真时刻 Tm 为 0，调试模块 Gain

图5.53 逐模块执行系统调试示意图

3. 逐时间步执行调试

逐模块调试系统可使用户对系统中任何模块的输入输出进行详细的观测，但是此调试方式非常耗时，尤其是对大型复杂系统进行调试。Simulink允许用户使用逐时间步方式对系统进行调试。逐时间步调试的特点是调试的时间间隔与系统仿真步长一致，并且在系统断点处停止执行。图5.54为逐时间步执行系统调试的示意图。

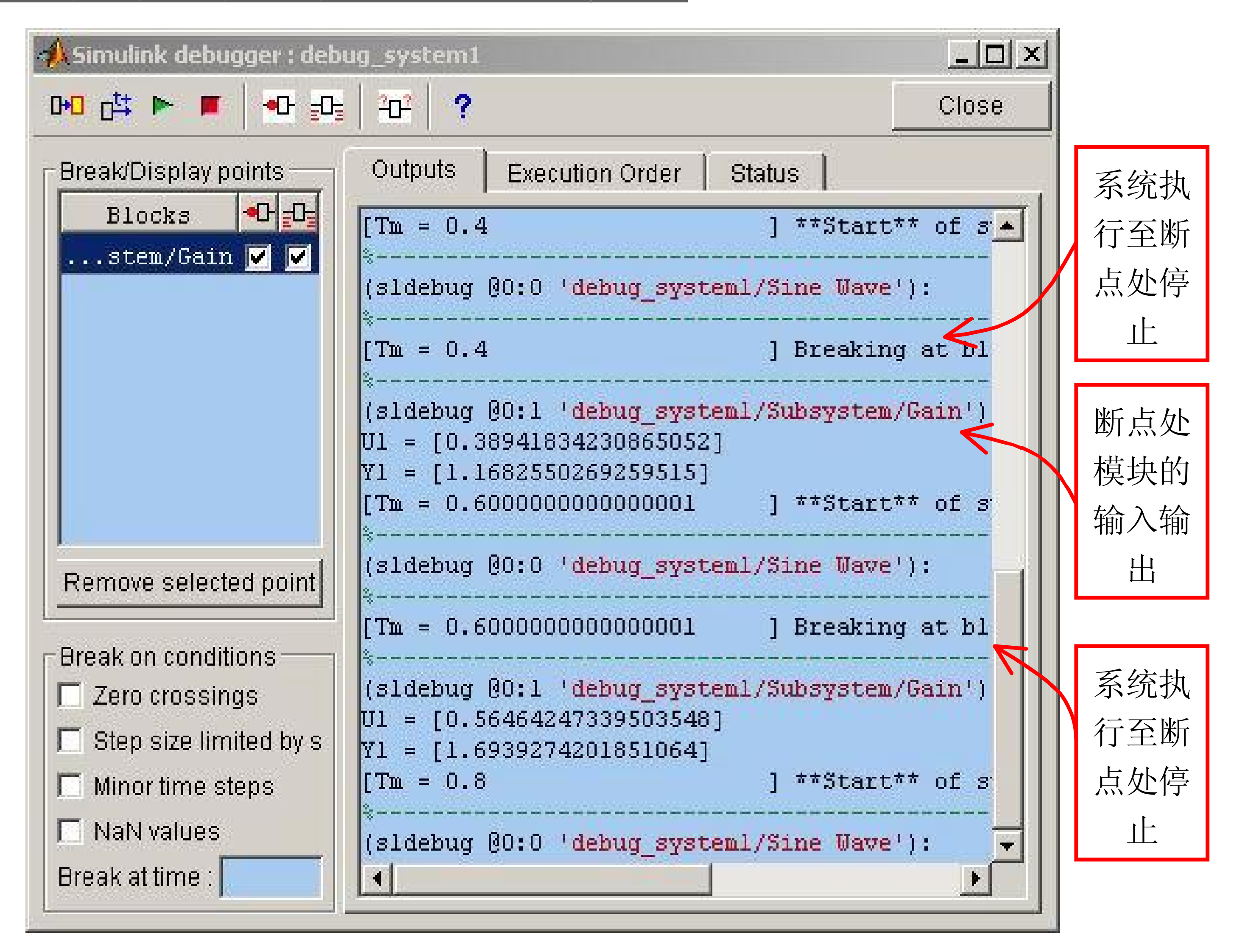


图5.54 逐时间执行系统调试示意图

4．调试执行顺序与调试状态

在系统调试过程中，Simulink调试器按照一定的顺序对系统模块进行调试。使用调试器输出窗口中的Execution Order窗口可以显示系统模块的调试顺序。此功能对于简单的系统调试而言作用不大，但对于大型多速率复杂系统来说是非常重要的。另外，如果系统中包含代数环，在Execution Order窗口中也会显示相关的系统模块。

Simulink debugger : debug_system1

Close

Break/Display points

Blocks

...stem/Gain

Remove selected point

Break on conditions

Zero crossings

Step size limited by

Minor time steps

NaN values

Break at time :

Outputs | Execution Order | Status

```
---- Sorted list for 'debug_system1' [9 bloc
  0:0    'debug_system1/Sine Wave' (Sin)
  0:1    'debug_system1/Subsystem/Gain' (Gai
  0:2    'debug_system1/Scope' (Scope)
  0:3    'debug_system1/Scope1' (Scope)
```

模块调试顺序

```
Current simulation time
Default command to execute on return/enter
Stop in minor times steps
Break at zero crossing events
Break when step size is limiting by a stat
Time break point
Break on non-finite (NaN,Inf) values
Number of installed break points
  Break point 0:1 'debug_system1/Subsystem
Number of installed display points
Number of installed trace points
  Trace point 0:1 'debug_system1/Subsystem
Display items for disp, trace, probe
Display of integration information
Algebraic loop tracing level
Window reuse
Execution Mode
```

系统调试状态

图5.55 系统调试中模块的执行顺序与调试状态



第6章　Simulink系统仿真原理



6.1 Simulink求解器概念

6.1.1 离散求解器

第3章中简单介绍了动态系统的模型及其描述，其中指出，离散系统的动态行为一般可以由差分方程描述。众所周知，离散系统的输入与输出仅在离散的时刻上取值，系统状态每隔固定的时间才更新一次；而Simulink对离散系统的仿真核心是对离散系统差分方程的求解。

在对纯粹的离散系统进行仿真时，需要选择离散求解器对其进行求解。用户只需选择Simulink仿真参数设置对话框中的求解器选项卡中的discrete（no continuous states）选项，即没有连续状态的离散求解器，便可以对离散系统进行精确的求解与仿真。读者可以参考第5章中相关内容了解离散求解器的其它设置，这里不再赘述。

6.1.2 连续求解器

与离散系统不同，连续系统具有连续的输入与输出，并且系统中一般都存在着连续的状态变量。连续系统中存在的状态变量往往是系统中某些信号的微分或积分，因此连续系统一般由微分方程或与之等价的其它方式进行描述。这就决定了使用数字计算机不可能得到连续系统的精确解，而只能得到系统的数字解（即近似解）。

采用不同的连续求解器会对连续系统的仿真结果与仿真速度产生不同的影响，但一般不会对系统的性能分析产生较大的影响，因为用户可以设置具有一定的误差范围的连续求解器进行相应的控制。离散求解器与连续求解器设置的不同之处如图6.1所示。

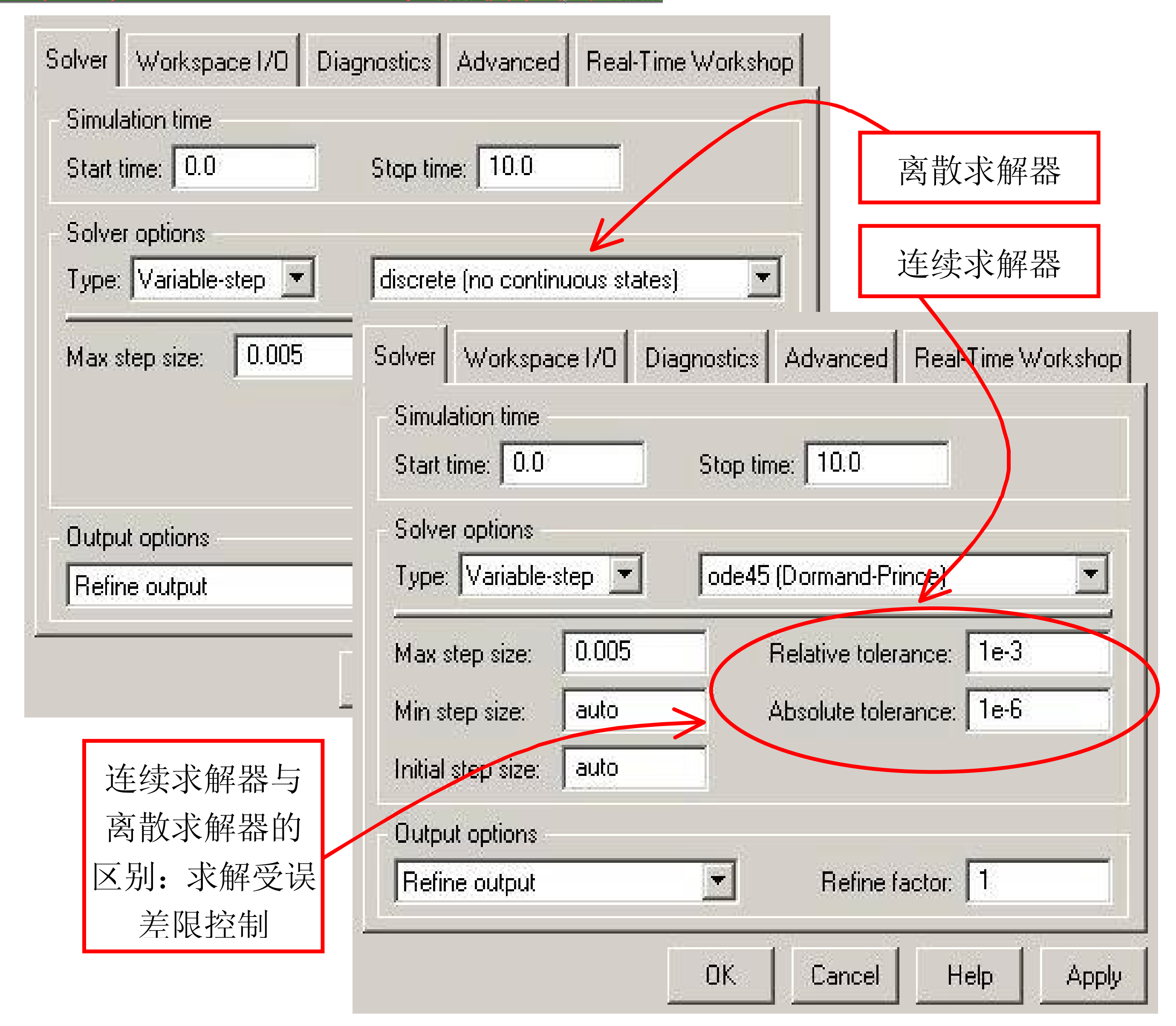


图6.1 离散求解器与连续求解器设置的比较

由于连续系统状态变量不能够被精确地计算出来，因而积分的误差值同样也是一个近似值。通常，连续求解器采用两个不同阶次的近似方法进行积分，然后计算它们之间的积分差值作为积分误差。连续求解器积分误差的计算如图6.2所示。

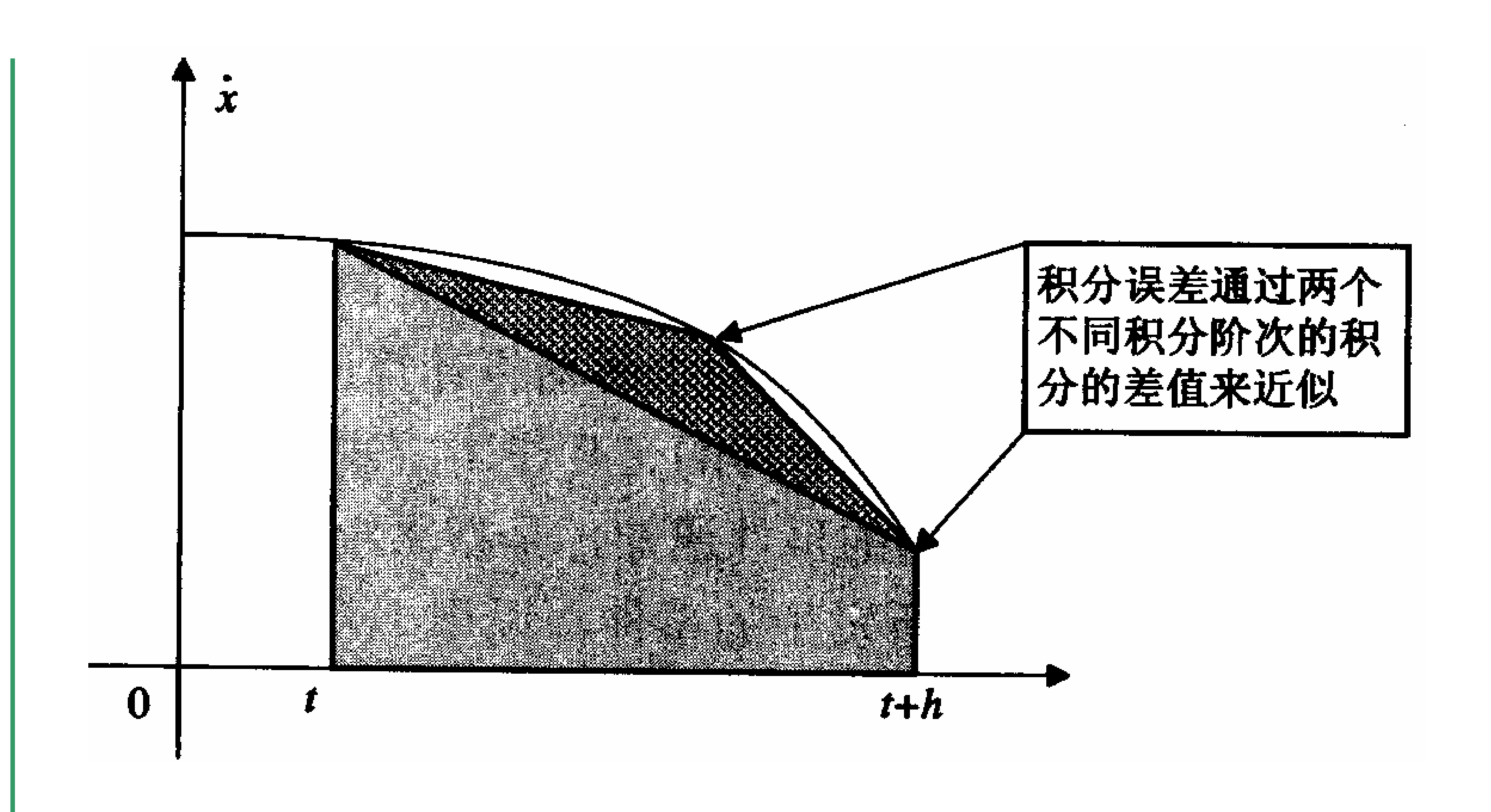


图6.2 连续求解器积分误差计算

图6.2中h为积分步长。注意，此图以最简单的多边形积分近似算法为例说明积分误差的计算，在实际中具体的方法视连续求解器的不同而不同。如果积分误差满足绝对误差或相对误差，则仿真继续进行；如果不满足，则求解器尝试一个更小的步长，并重复这个过程。当然，连续求解器在选择更小步长时采用的方法也不尽相同。如果误差上限值的选择或连续求解器的选择不适合待求解的连续系统，则仿真步长有可能会变得非常小，使仿真速度变得非常慢。(用户需要注意这一点。)

混合系统仿真时连续状态求解与离散状态求解的协调如图6.3所示。其中h为初始步长，由于在时刻t与$t+h$之间系统存在着离散状态的更新，因而连续变步长求解器将会减小步长至，之后再计算积分误差以控制求解。如果求解误差满足误差范围，则进行下一步仿真，否则缩小时间间隔，重复此过程进行求解仿真。

$\dot{x}$

h_{new}

0　t　$t+h$

离散更新时刻

图6.3 连续状态求解与离散状态求解的协调示意图



6.2 系统过零的概念与解决方案

6.1节中对Simulink的求解器进行了较为深入的介绍。Simulink求解器固然是系统仿真的核心，但Simulink对动态系统求解仿真的控制流程也是非常关键的。Simulink对系统仿真的控制是通过系统模型与求解器之间建立对话的方式进行的：Simulink将系统模型、模块参数与系统方程传递给Simulink的求解器，而求解器将计算出的系统状态与仿真时间通过Simulink环境传递给系统模型本身，通过这样的交互作用方式来完成动态系统的仿真。

6.2.1 过零的产生

在动态系统的仿真过程中，所谓过零，是指系统模型中的信号或系统模块特征的某种改变。这种特征改变包括以下两种情况：

(1) 信号在上一个仿真时间步长之内改变了符号。

(2) 系统模块在上一个仿真时间步长改变了模式（如积分器进入了饱和区段）。

6.2.2 事件通知

在动态系统仿真中，采用变步长求解器可以使Simulink正确地检测到系统模块与信号中过零事件的发生。当一个模块通过Simulink仿真环境通知求解器，在系统前一仿真步长时间内发生了过零事件，变步长求解器就会缩小仿真步长，即使求解误差满足绝对误差和相对误差的上限要求。缩小仿真步长的目的是判定事件发生的准确时间（也就是过零事件发生的准确时刻）。

6.2.3 支持过零的模块

在Simulink的模块库中，并非所有的模块都能够产生过零事件。

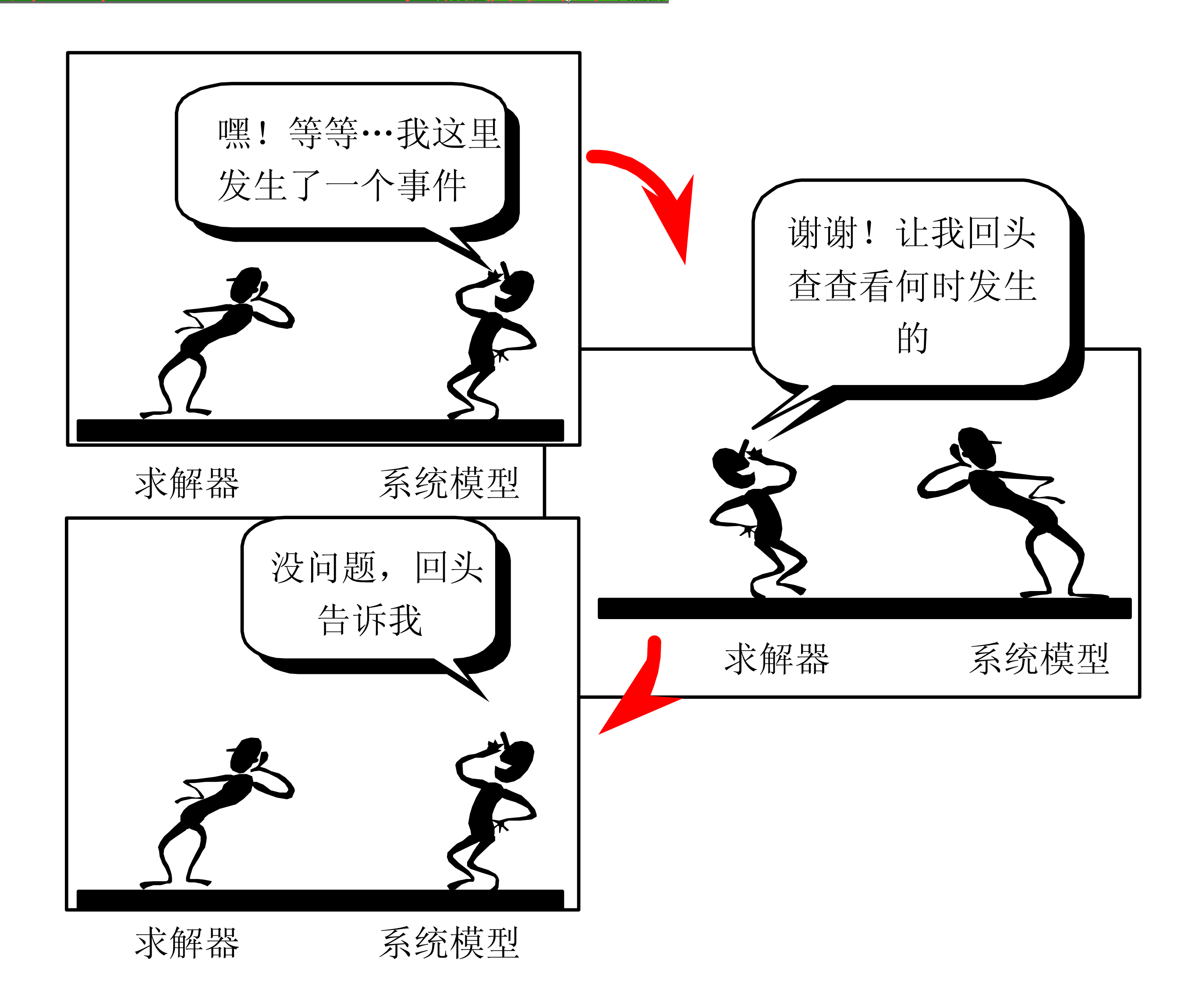
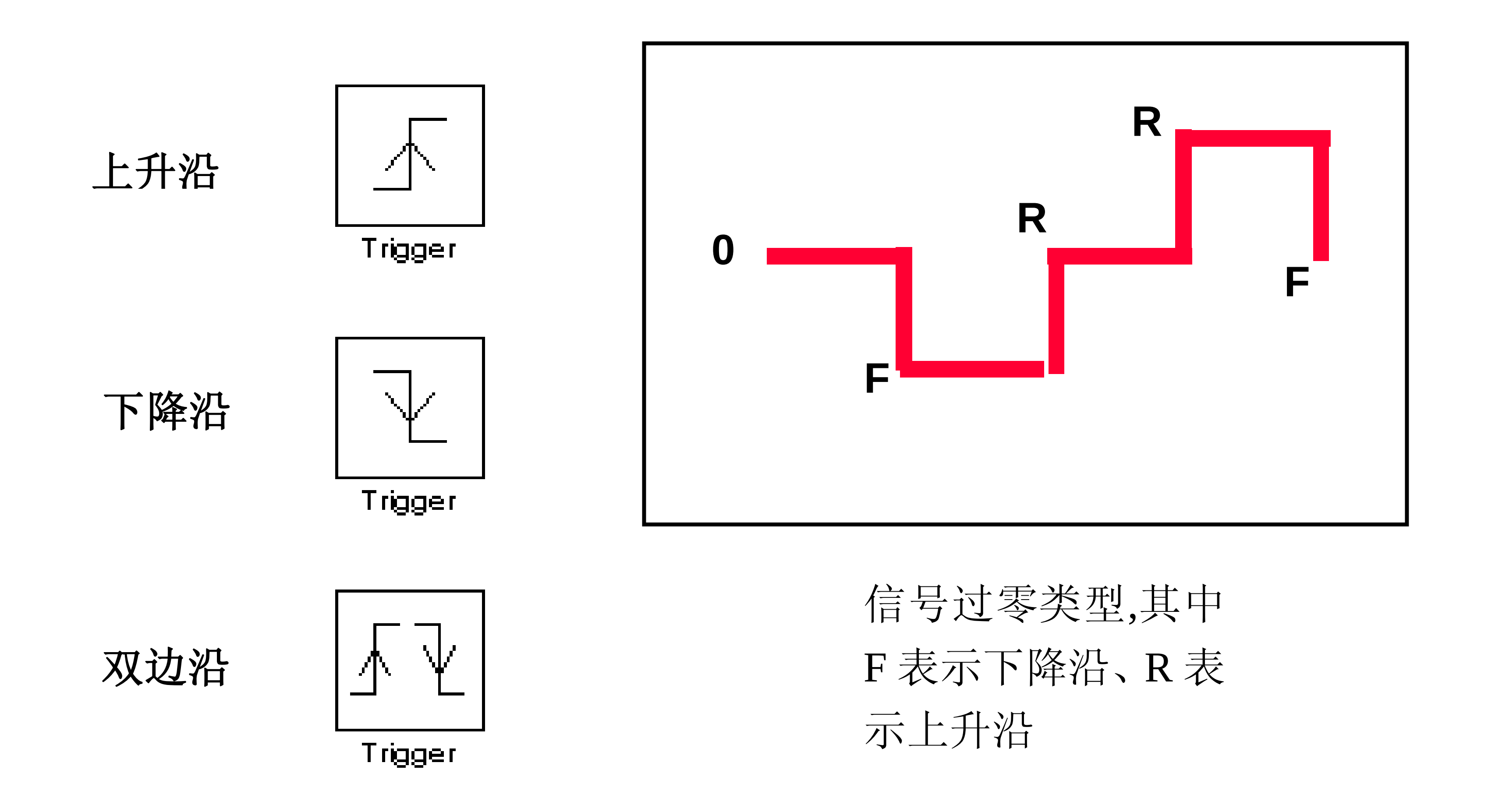


图6.4 系统模型与求解器之间的交互作用示意图

对于其它的许多模块而言，它们不具有过零检测的能力。如果需要对这些模块进行过零检测，则可以使用信号与系统库（Signals & Systems）中的Hit Crossing零交叉模块来实现。当Hit Crossing模块的输入穿过某一偏移值（offset）时会产生一个过零事件，所以它可以用来为不带过零能力的模块提供过零检测的能力。

一般而言，系统模型中模块过零的作用有两种类型：一是用来通知求解器，系统的运行模式是否发生了改变，也就是系统的动态特性是否发生改变；二是驱动系统模型中其它模块。过零信号包含三种类型：上升沿、下降沿、双边沿，如图6.5所示。



信号过零类型,其中F表示下降沿、R表示上升沿

图6.5 过零信号的类型

下面分别对这三种类型进行简单的介绍。

(1) 上升沿：系统中的信号上升到零或穿过零，或者是信号由零变为正。

(2) 下降沿：系统中的信号下降到零或穿过零，或者是信号由零变为负。

(3) 双边沿：任何信号的上升或下降沿的发生。

6.2.4 过零的举例——过零的产生与关闭过零

1. 过零点的产生

【例6.1】 过零的产生与影响。

这里以一个很简单的例子来说明系统中过零的概念以及它对系统仿真所造成的影响。在这个例子中，采用Functions & Tables-函数与表库中的Function函数模块和Math数学库中的Abs绝对值模块分别计算对应输入的绝对值。我们知道，Function模块不会产生过零事件，所以在求取绝对值时，一些拐角点被漏掉了；但是Abs模块能够产生过零事件，所以每当它的输入信号改变符号时，它都能够精确地得到零点结果。图6.6所示为此系统的Simulink模型以及系统仿真结果。

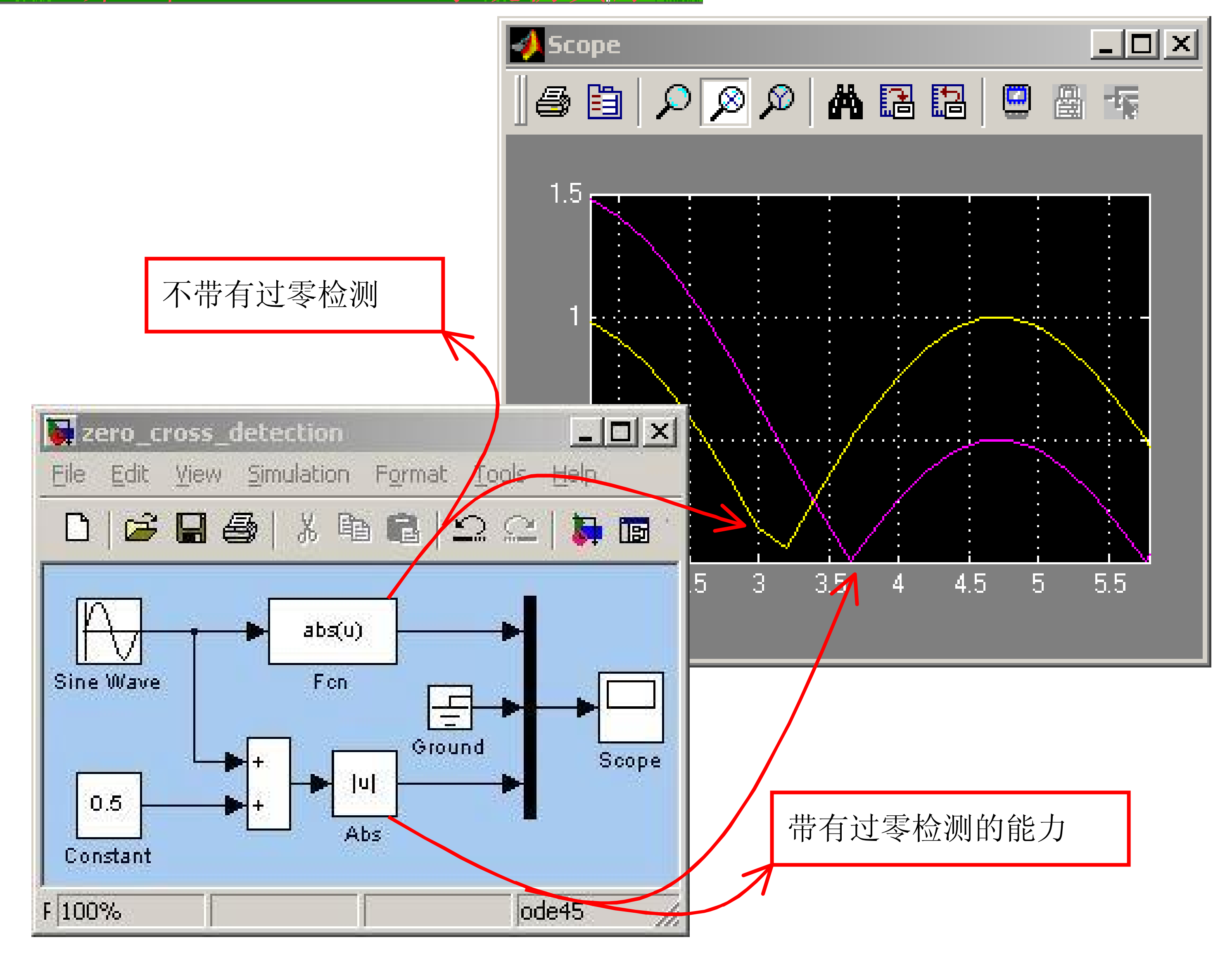

不带有过零检测
Scope
1.5
1
3
3.5
4
4.5
5
5.5
zero_cross_detection
File Edit View Simulation Format Tools Help
abs(u)
Fcn
Sine Wave
Ground
Scope
|u|
Abs
0.5
Constant
100%
ode45
带有过零检测的能力

图6.6 过零产生的影响

从仿真的结果中可以明显地看出，对于不带有过零检测的Function函数模块，在求取输入信号的绝对值时，漏掉了信号的过零点（即结果中的拐角点）；而对于具有过零检测能力的Abs求取绝对值模块，它可以使仿真在过零点处的仿真步长足够小，从而可以获得精确的结果。为说明这一点，在MATLAB命令窗口中输入如下语句：

```
>> semilogy(tout(1:end–1,diff(tout))
```

% 绘制系统仿真时刻的一阶差分（即系统仿真步长），如图6.7所示，其中常规步长为0.2 s，

% 当发生过零的情况时，系统仿真步长自动缩小至约s

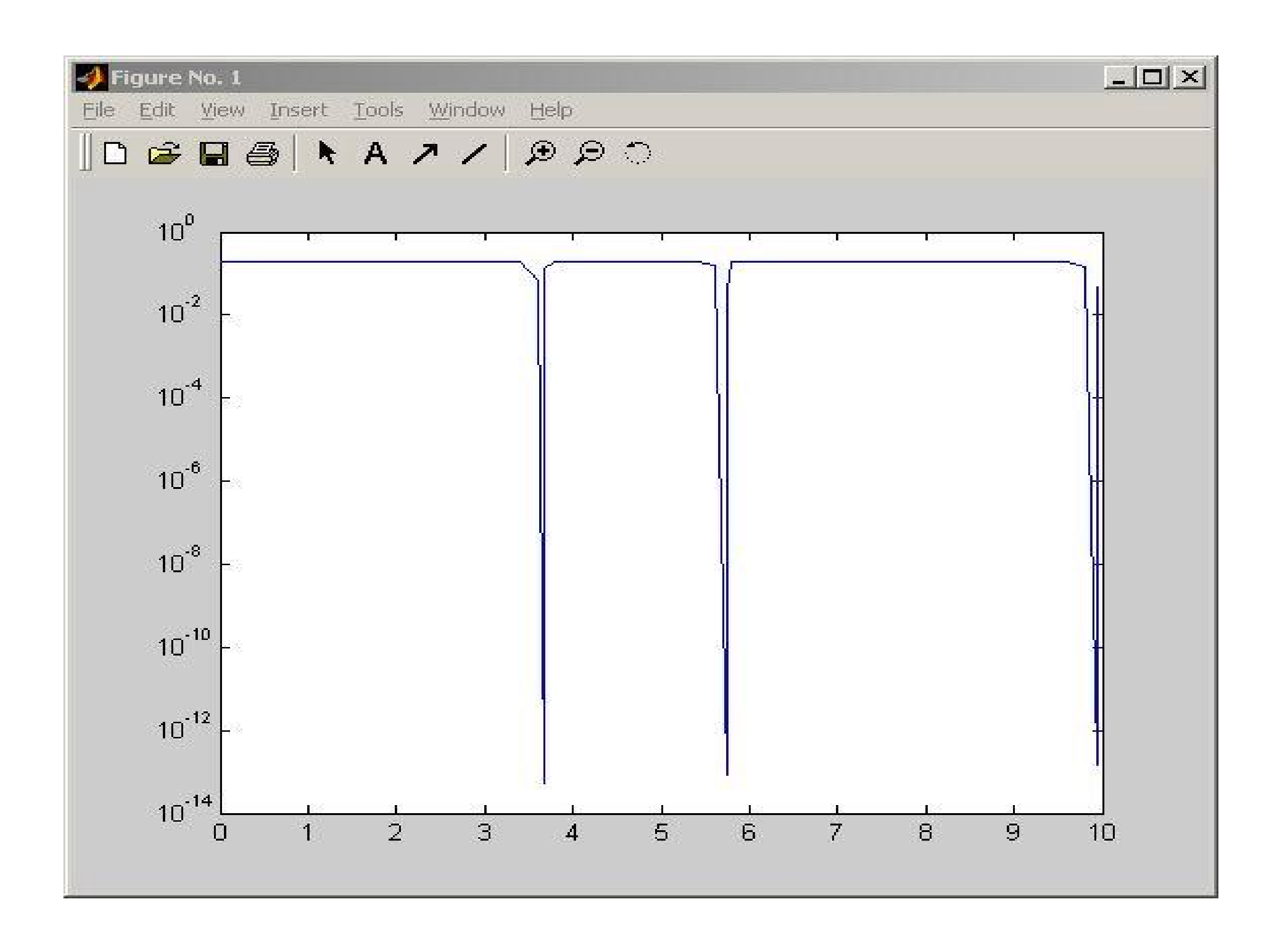
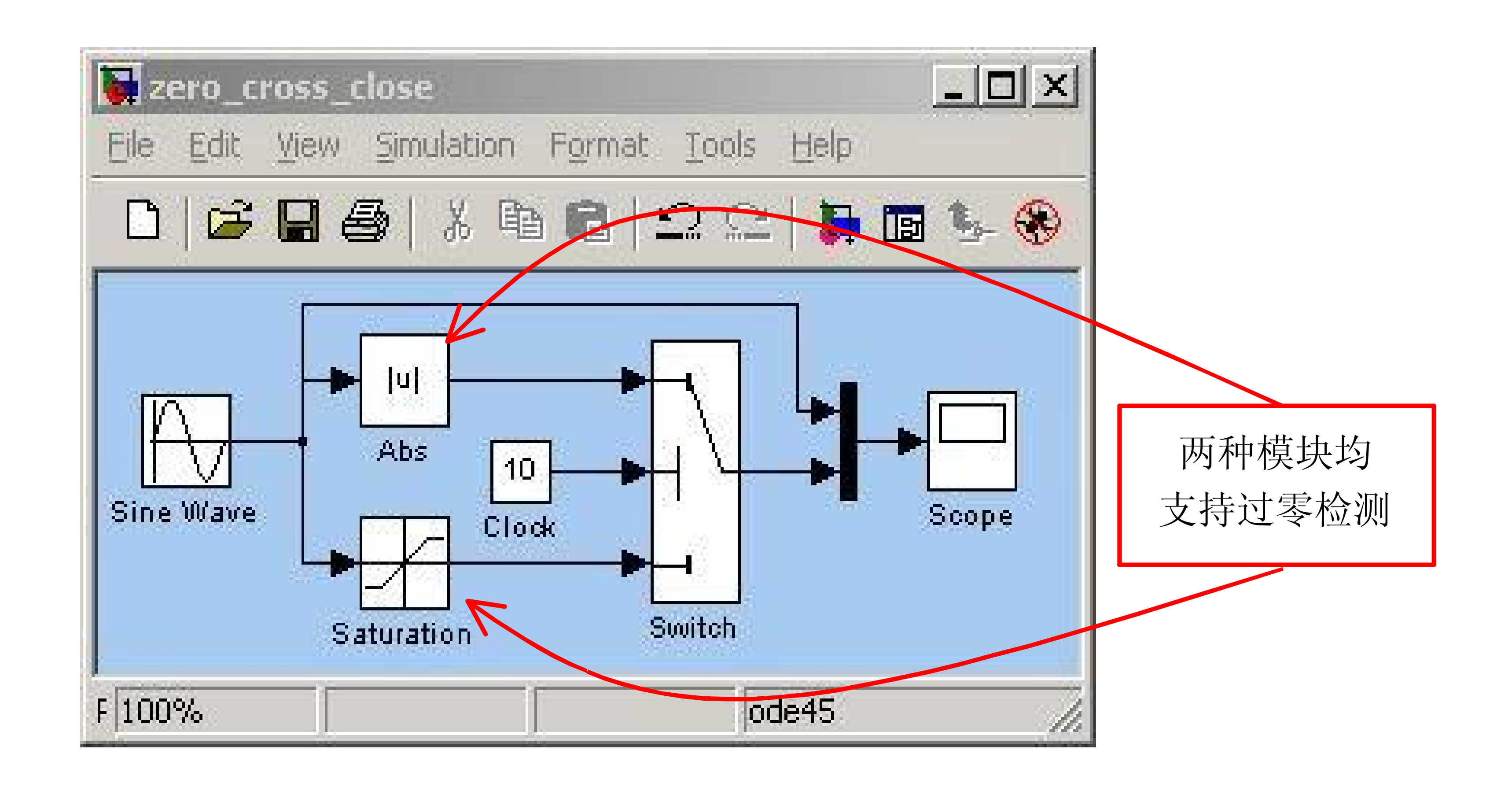


图6.7 系统仿真中过零处步长变化

2. 关闭过零

【例6.2】 过零的关闭与影响。

在【例6.1】中，过零表示系统中信号穿过了零点。其实，过零不仅用来表示信号穿过了零点，还可以用来表示信号的陡沿和饱和。在这个例子中，系统实现了输入信号由其绝对值跳变到饱和值的功能，而且跳变过程受到仿真时刻的控制。在此系统模型中所使用的Abs模块与Saturation模块都支持过零事件的产生，因此在系统的响应输出中得到了理想的陡沿。其中系统模型如图6.8(a)所示，系统仿真结果如图6.8(b)所示。


zero_cross_close
File Edit View Simulation Format Tools Help
|u|
Abs
Sine Wave
10
Clock
Saturation
Switch
Scope
100%
ode45
两种模块均
支持过零检测

Scope

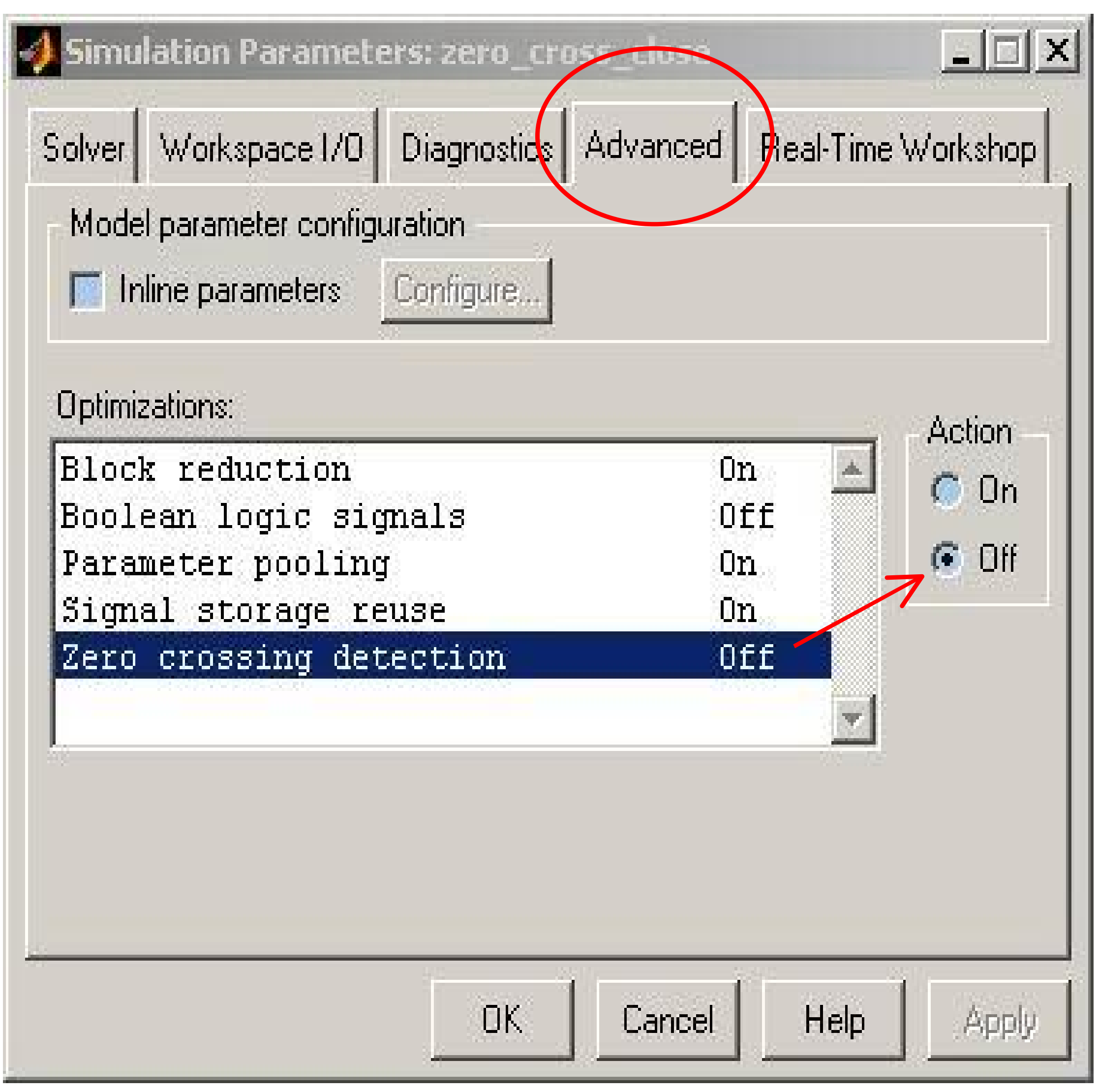
1
0.5
0
-0.5
-1
0 2 4 6 8 10
Time offset: 0

使用过零检测可以获得很好的仿真结果

图6.8 系统模型及系统仿真结果

从图6.8中可以明显看出，使用过零检测可以获得很好的仿真结果，系统的输出具有很好的陡沿。

在使用Simulink进行动态系统仿真中，其默认参数选择使用过零检测的功能。如果使用过零检测并不能给系统的仿真带来很大的好处，用户可以关闭仿真过程中过零事件的检测功能。用户可以在Simulation Parameters 参数设置对话框中的Advanced选项卡中进行设置，以关闭过零检测功能，然后再次对系统进行仿真。图6.9(a)、(b)所示分别为关闭过零检测的设置以及在关闭过零检测后系统的仿真结果。


Simulation Parameters: zero_cross_close
Solver
Workspace I/O
Diagnostics
Advanced
Real-Time Workshop
Model parameter configuration
Inline parameters
Configure...
Optimizations:
Block reduction On
Boolean logic signals Off
Parameter pooling On
Signal storage reuse On
Zero crossing detection Off
Action
On
Off
OK
Cancel
Help
Apply

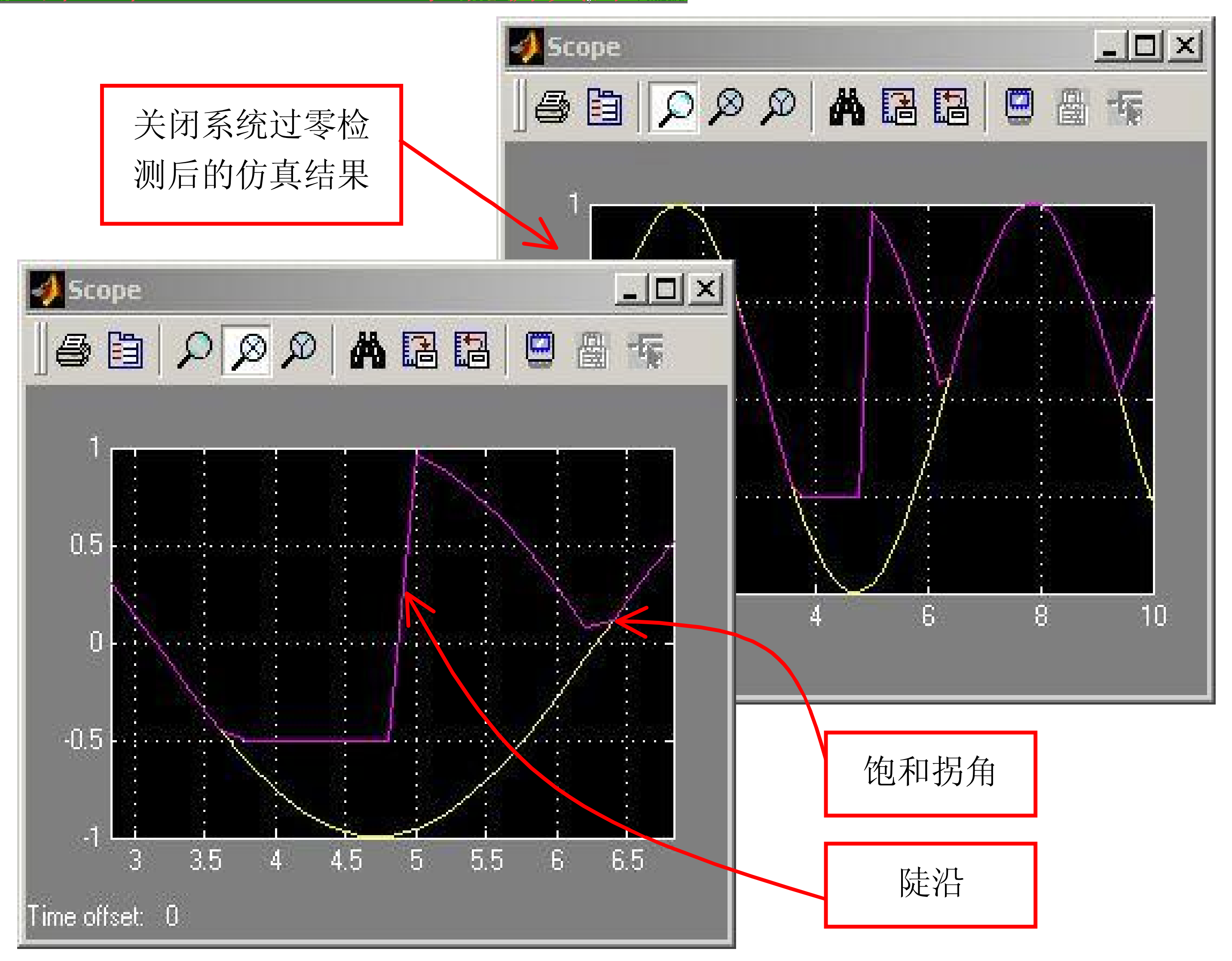


图6.9 关闭系统过零检测的设置和关闭过零检测后的仿真结果

6.2.5 使用过零检测的其它注意事项

在使用过零检测时，用户需要注意如下几点：

(1) 关闭系统仿真参数设置中的过零事件检测，可以使动态系统的仿真速度得到很大的提高。但可能会引起系统仿真结果的不精确，甚至出现错误结果。

(2) 关闭系统过零检测对Hit Crossing零交叉模块并无影响。

(3) 对于离散模块及其产生的离散信号不需要进行过零检测。



6.3 系统代数环的概念与解决方案

6.3.1直接馈通模块

在使用Simulink的模块库建立动态系统的模型时，有些系统模块的输入端口（Input ports）具有直接馈通（Direct feedthrough）的特性。所谓模块的直接馈通，是指如果在这些模块的输入端口中没有输入信号，则无法计算此模块的输出信号。换句话说，直接馈通就是模块输出直接依赖于模块的输入。在Simulink中具有直接馈通特性的模块有如下的几种：

(1) Math Function数学函数模块。

(2) Gain增益模块。

(3) Product乘法模块。

(4) State-Space状态空间模块（其中矩阵***D***不为0）。

(5) Transfer Fcn传递函数模块（分子与分母多项式阶次相同）。

(6) Sum求和模块。

(7) Zero-Pole零极点模块（零点与极点数目相同）。

(8) Integrator积分模块。

6.3.2 代数环的产生

在介绍完具有直接馈通特性的系统模块之后，来介绍代数环的产生。系统模型中产生代数环的条件如下：

(1) 具有直接馈通特性的系统模块的输入，直接由此模块的输出来驱动。

(2) 具有直接馈通特性的系统模块的输入，由其它直接馈通模块所构成的反馈回路间接来驱动。

图6.10所示为一个非常简单的标量代数环的构成。

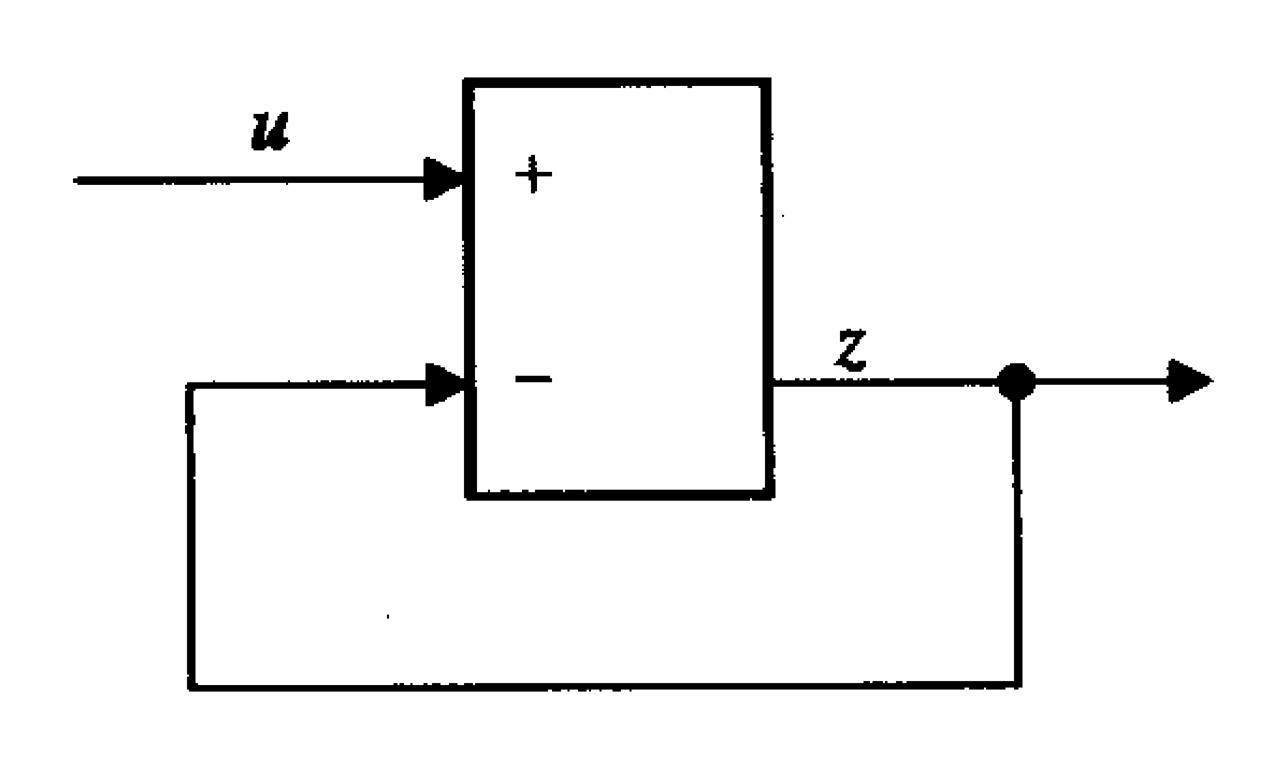


图6.10 标量代数环

6.3.3 代数环的举例与解决方案之一：直接求解系统方程

【例6.3】 代数环的直接求解。在图6.11中所示的两个系统模型中均存在代数环结构，试对这两个系统进行求解

解： 为了计算求和模块Sum的输出，需要知道其输入，但是其输入恰恰包含模块的输出。对于此二系统，很容易写出如下所示的系统的动态方程：

(1) 第一个系统模型的动态方程：，所以。

(2) 第二个系统模型的动态方程：，所以。

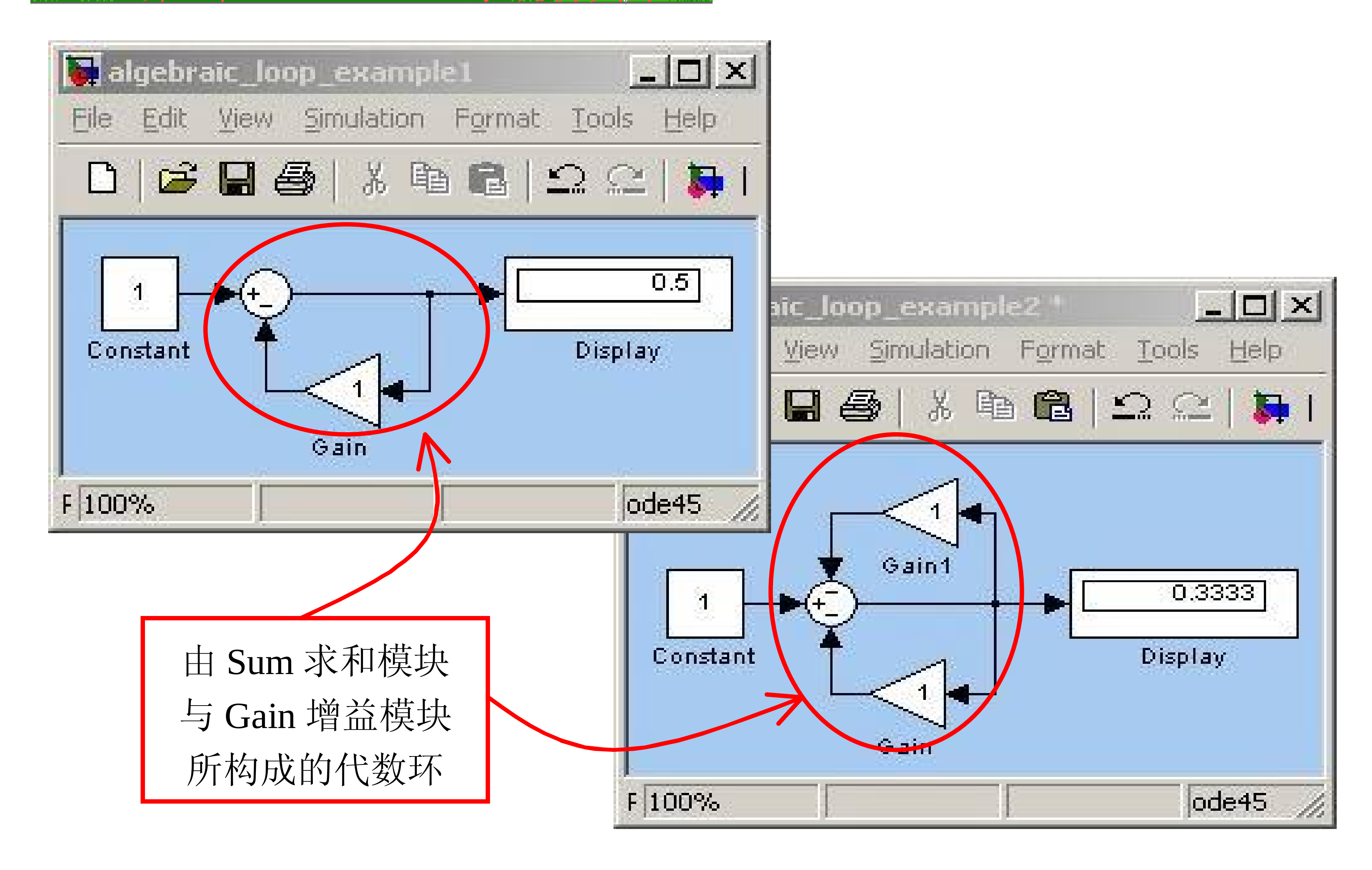


图6.11 具有代数环的系统模型

6.3.4 代数环的举例与解决方案之二：代数约束

用户除了可以使用Simulink内置的代数环求解器对含有代数环的动态系统进行仿真，还可以使用Math模块库中的代数约束Algebraic Constraint模块对动态系统数学方程进行求解。

使用代数约束模块并给出约束初始值，可以方便地对代数方程进行求解。代数约束模块通过调整其输出代数状态以使其输入为零。其中为模块的输出状态，为一代数表达式，它作为模块的输入。

【例6.4】确良 使用代数约束求解代数环。

在如图6.12所示的系统模型中代数约束模块的输出分别为代数状态。分别通过反馈回路作为代数约束模块的输入。运行此系统相当于对如下的代数方程进行求解：

$$\begin{cases} z1 + z2 - 1 = 0 \\ z2 - z1 - 1 = 0 \end{cases}$$

其仿真结果如图6.12中Display模块所显示的那样，其中，。

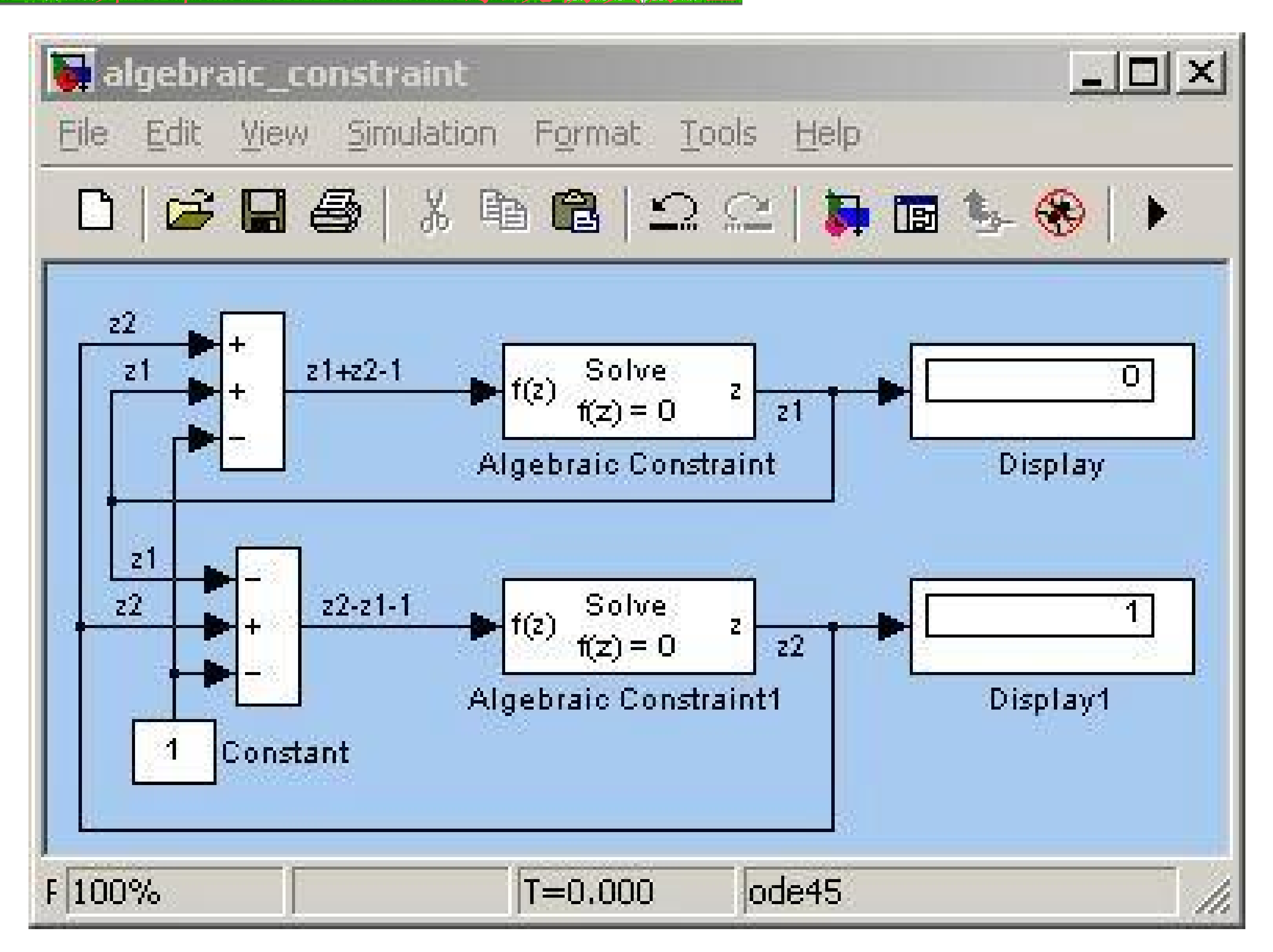


图6.12　使用代数约束求解的代数环结构

【例6.5】代数状态的初始值选取。使用代数约束来求解方程

$x^2 - x - 2 = 0 \qquad (x+1)(x-2) = 0$

的根（显然此方程的根为 $x_1 = -1$ $x_2 = 2$ ）

解：首先建立如图6.13所示的系统模型，然后对代数约束模块的初始值进行设置，如图6.13所示（仿真结果如Display模块中所示）。

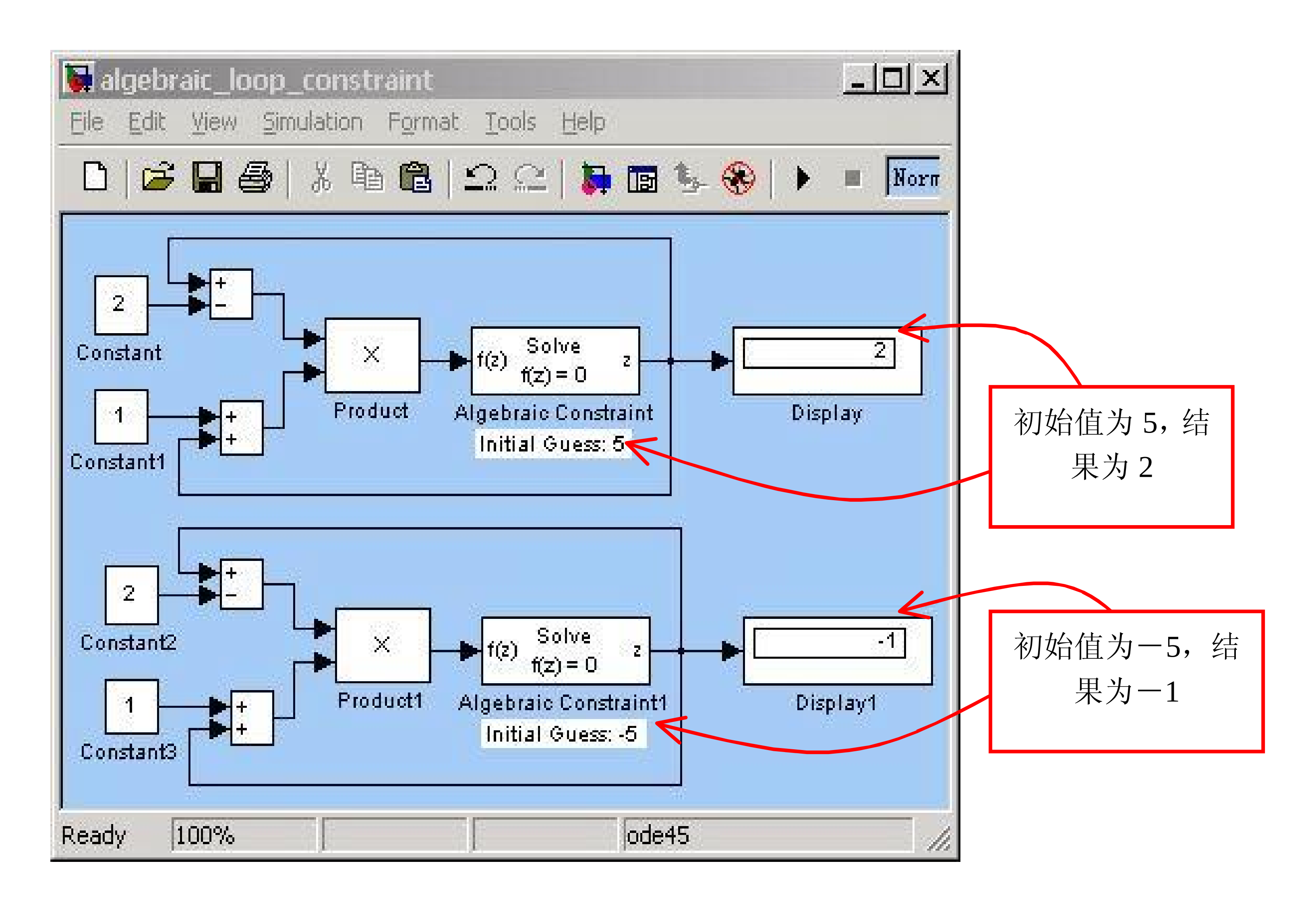


图6.13　代数状态的初始值选择

6.3.5 代数环的举例与解决方案之三：切断环

至此，读者能够采用两种方法对含有代数环的动态系统进行仿真分析：一是直接对系统方程进行手工求解，但是在很多情况下难以进行手工求解甚至不可能进行手工求解；二是使用代数约束，由Simulink内置的代数环求解器对含有代数环的系统进行仿真。

【例6.6】 对于如下的连续线性系统：

$$G(s)=\frac{s+0.5}{(s-2)(s+3)}$$

建立如图6.14所示的系统模型。

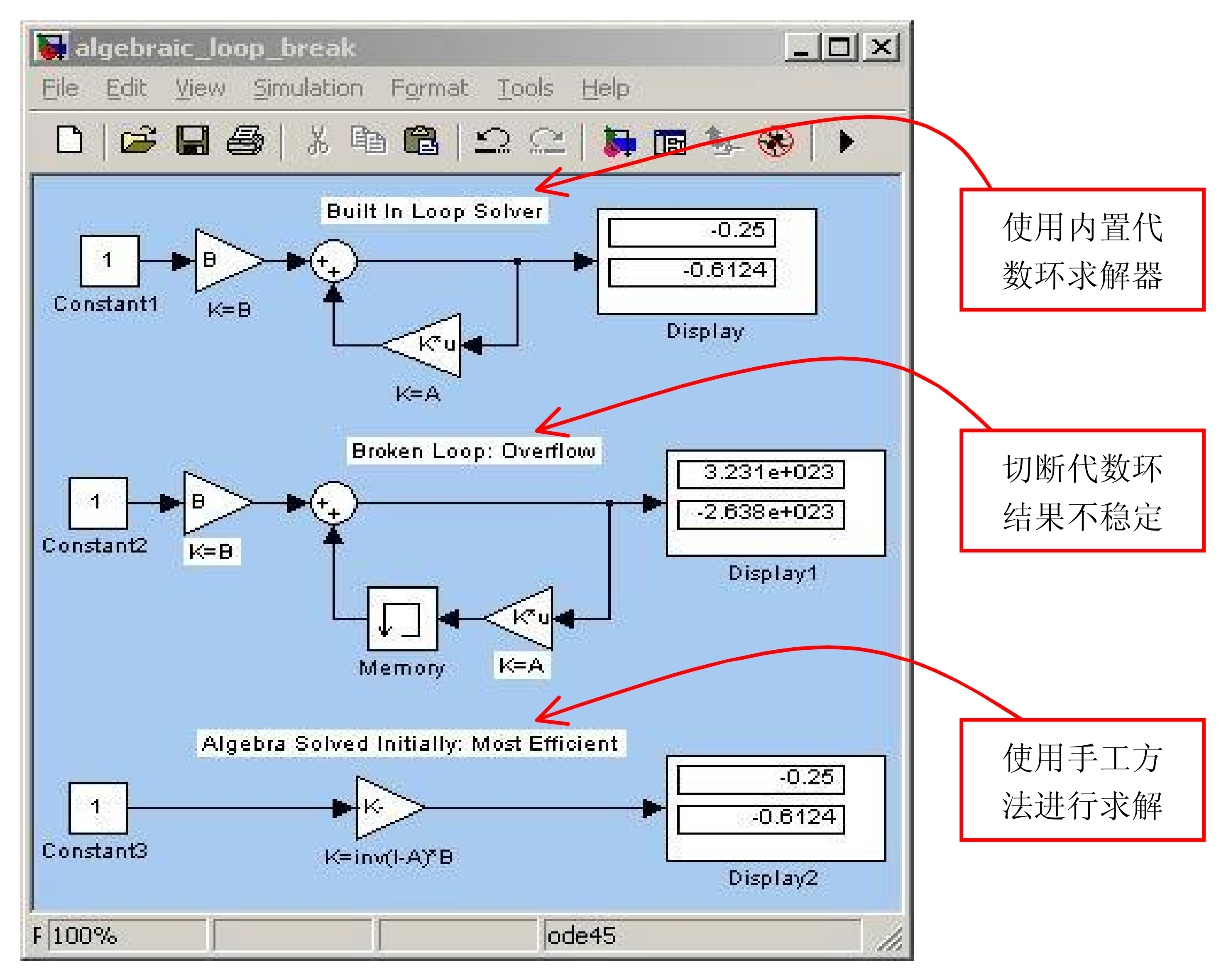

algebraic_loop_break
File Edit View Simulation Format Tools Help
Built In Loop Solver
1
Constant1
B
K=B
K*u
K=A
-0.25
-0.6124
Display
Broken Loop: Overflow
1
Constant2
B
K=B
Memory
K*u
K=A
3.231e+023
-2.638e+023
Display1
Algebra Solved Initially: Most Efficient
1
Constant3
K-
K=inv(I-A)*B
-0.25
-0.6124
Display2
100%
ode45
使用内置代数环求解器
切断代数环结果不稳定
使用手工方法进行求解

图6.14 切断代数环

图6.14中为线性连续系统的零极点描述，其相应的状态空间描述矩阵分别为***A***，***B***，***C***，***D***。建立此系统的模型以求出系统在恒定输入下的状态值。为了说明切断代数环的影响，我们在此模型中给出系统状态求取的三种不同方法以作为比较。图中最上方为使用Simulink的内置代数环求解器进行状态求解，最下方为使用手工方式进行状态求解，中间为使用Memory模块切断代数环，然后进行状态求解。



6.4　高级积分器

在使用Simulink对实际的动态系统进行仿真时，积分运算可以说是Simulink求解器的核心技术之一。前面在介绍动态系统的仿真实现时，仅仅使用了最简单的积分器设置。在这一节中，将简单介绍高级积分器的概念及其应用。

首先对积分器的各个端口进行简单的介绍。图6.15所示为使用缺省参数设置下的积分器外观与选择所有参数设置后积分器的外观比较。

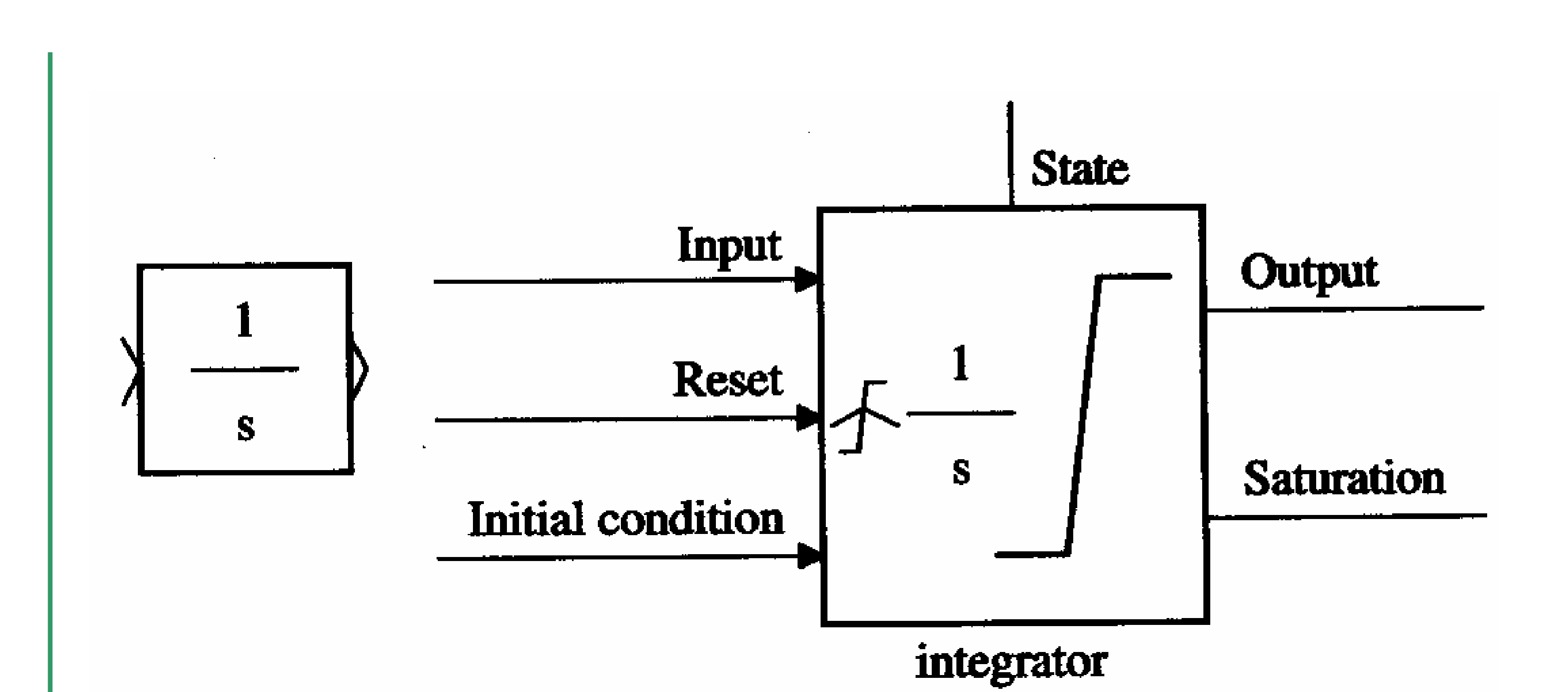


图6.15 积分器外观比较

对于使用缺省参数设置下的积分器，其输出信号为输入信号的数值积分，想必读者对其已经比较熟悉了，这里不再赘述。下面详细介绍一下选择所有参数设置后的积分器各个端口的含义以及对积分器的设置。首先介绍一下积分器参数设置对话框，如图6.16所示。

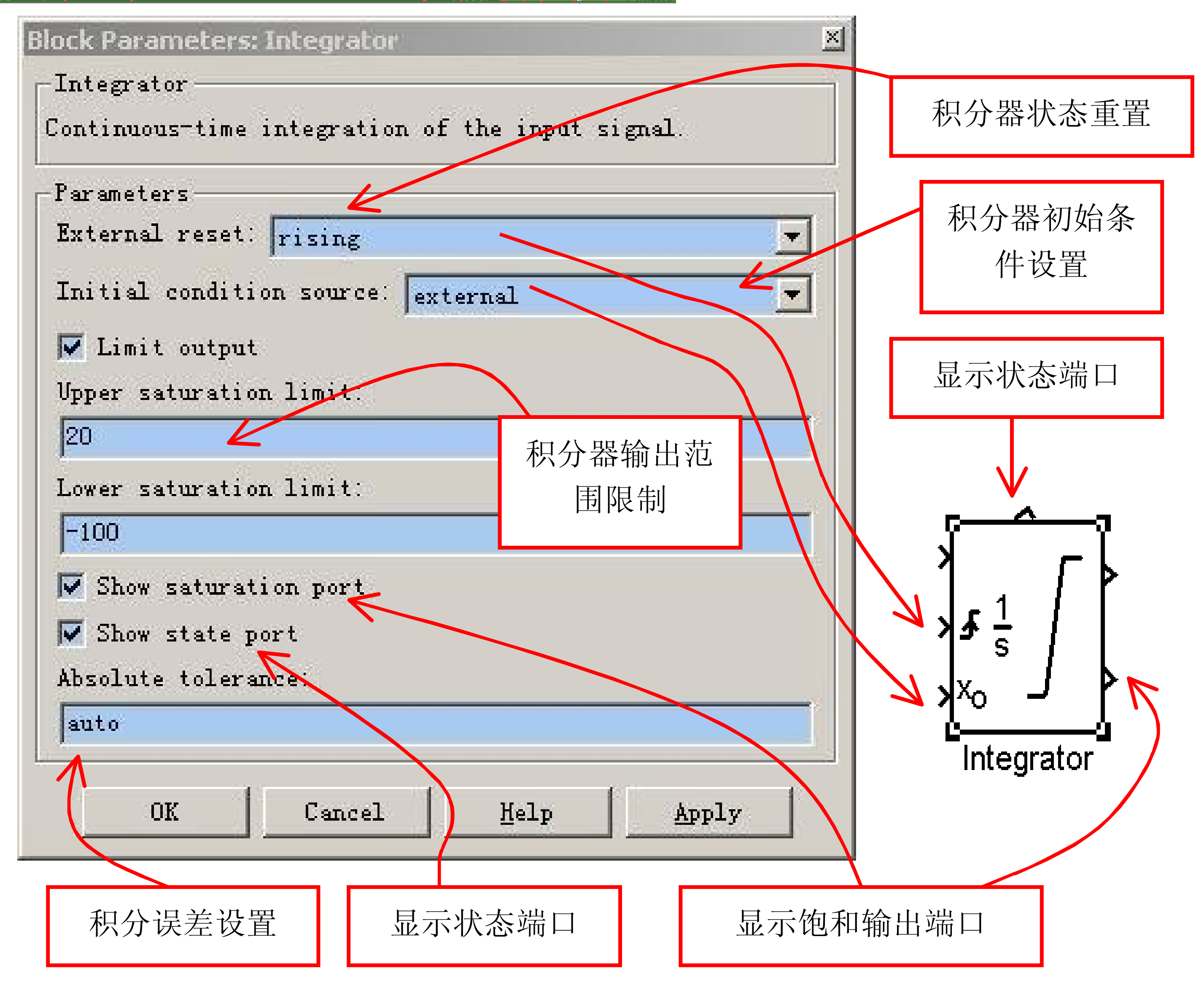


图6.16 高级积分器设置

1. 积分器的初始条件端口（Initial condition）

设置积分器初始条件的方法有两种，它们分别是：

(1) 在积分器模块参数设置对话框中设置初始条件：在初始条件源设置（Initial condition source）中选择内部设置（Internal），并在下面的文本框中键入给定的初始条件，此时不显示积分器端口。

(2) 从外部输入源设置积分器初始条件：在初始条件源设置中选择外部设置（External），初始条件设置端口以作为标志。此时需要使用Signals & Systems模块库中的IC模块设置积分器初始值。

2. 积分器状态端口（State）

当出现下述的两种情况时，需要使用积分器的状态端口而非其输出端口：

(1) 当积分器模块的输出经重置端口或初始条件端口反馈至模块本身时，会造成系统模型中出现代数环结构的问题，此时需要使用状态端口。

(2) 当从一个条件执行子系统向另外的条件执行子系统传递状态时，可能会引起时间同步问题。此时也需要使用状态端口而非输出端口。至于条件执行子系统的有关内容，将在第7章中介绍。

3. 积分器输出范围限制与饱和输出端口（Saturation）

在某些情况下，积分器的输出可能会超过系统本身所允许的上限或下限值，选择积分器输出范围限制框（Limit output），并设置上限值（Upper saturation limit）与下限值（Lower saturation limit），可以将积分器的输出限制在一个给定的范围之内。此时积分器的输出服从下面的规则：

(1) 当积分结果小于或等于下限值并且输入信号为负，积分器输出保持在下限值（下饱和区）。

(2) 当积分结果在上限值与下限值之间时，积分器输出为实际的积分结果。

(3) 当积分结果大于或等于上限值并且输入信号为正，积分器输出保持在上限值（上饱和区）。

选择Show saturation port复选框可以在积分器中显示饱和端口，此端口位于输出端口的下方。饱和端口的输出取值有三种情况，用来表示积分器的饱和状态：

(1) 输出为1，表示积分器处于上饱和区。

(2) 输出为0，表示积分器处于正常范围之内。

(3) 输出为-1，表示积分器处于下饱和区。

4. 积分器重置

选择积分器状态重置框可以重新设置积分器的状态，其值由外部输入信号决定。此时，在积分器输入端口下方出现重置触发端口。可以采用不同的触发方式对积分器状态进行重置：

(1) 当重置信号具有上升沿（rising）时，触发重置方式选择为上升沿。

(2) 当重置信号具有下降沿（falling）时，触发重置方式选择为下降沿。

(3) 当重置信号具有上升或下降沿（即双边沿）时，触发重置方式可选择为either。

(4) 当重置信号非零时，选择level重置积分器状态，并使积分器输出保持在初始状态。

积分器的重置端口具有直接馈通的特性。积分器模块的输出，无论是直接反馈还是通过具体直接馈通特性的模块反馈至其重置端口，都会使系统模型中出现代数环结构。而使用状态端口代替输出端口可以避免代数环的出现。

5. 积分器绝对误差设置（Absolute tolerance）

在默认情况下，积分器采用Simulink自动设置的绝对误差限。用户也可以根据自己的需要设置积分器的绝对误差限，在Absolute tolerance下键入误差上限即可。

至此，我们对高级积分器设置做了一个比较全面的介绍，下面举例说明。

【例6.7】 高级积分器的使用。系统模型如图6.17所示。

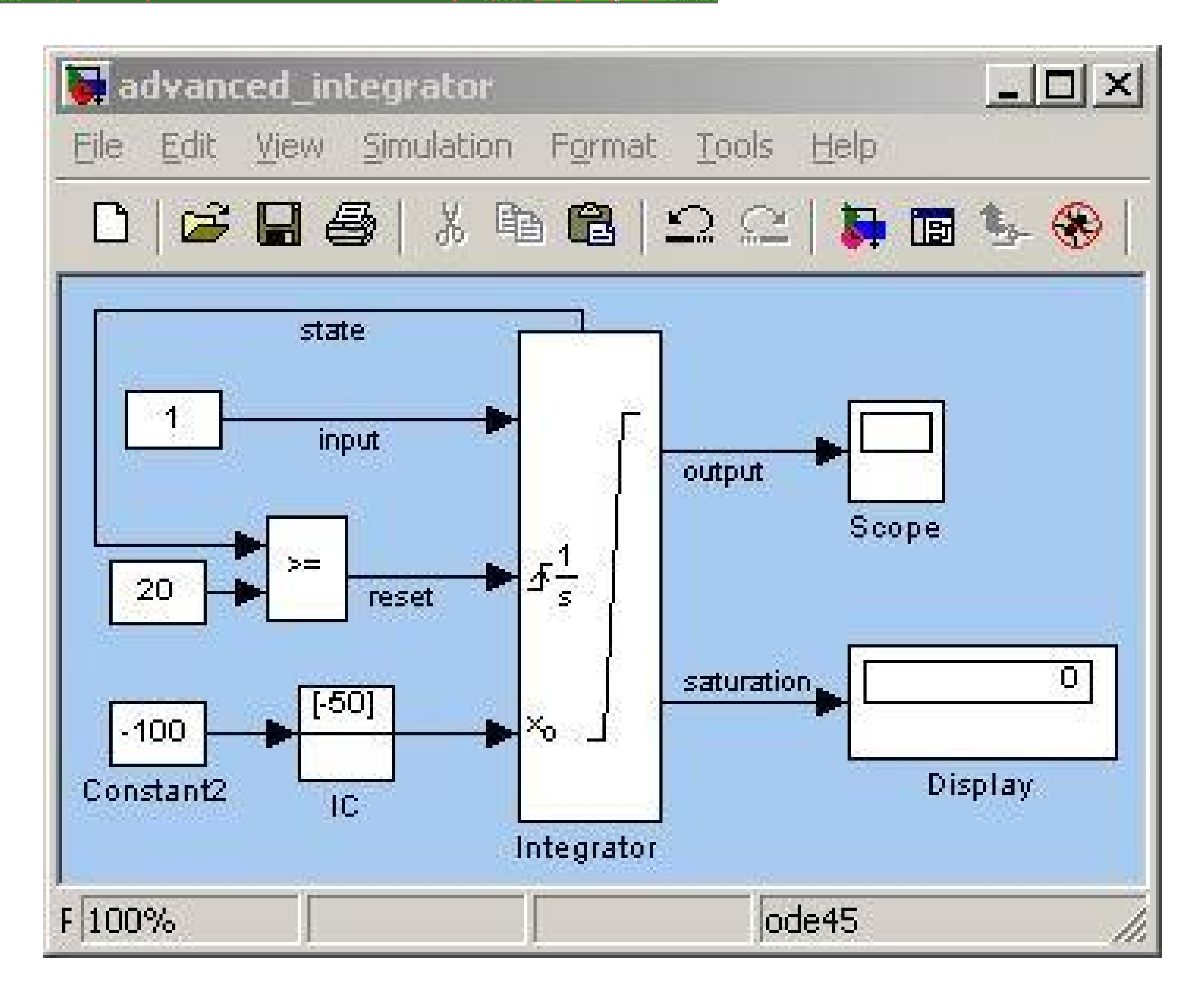


图6.17　高级积分器的使用

设置合适的仿真参数如下：仿真时间范围为0～200，积分器的输出上限为20，下限为，状态重置选择信号上升沿rising，其余参数如系统模型框图中所示。运行仿真结果如图6.18所示。从仿真结果中可以看出，在系统运行的初始时刻，积分器状态由IC模块所决定；而当系统的输出大于或等于20时，积分器状态重置为。

图6.18 系统仿真结果



6.5 仿真参数设置：高级选项与诊断选项

6.1.1 高级选项

首先介绍Simulink仿真参数的高级选项，如图6.19所示。

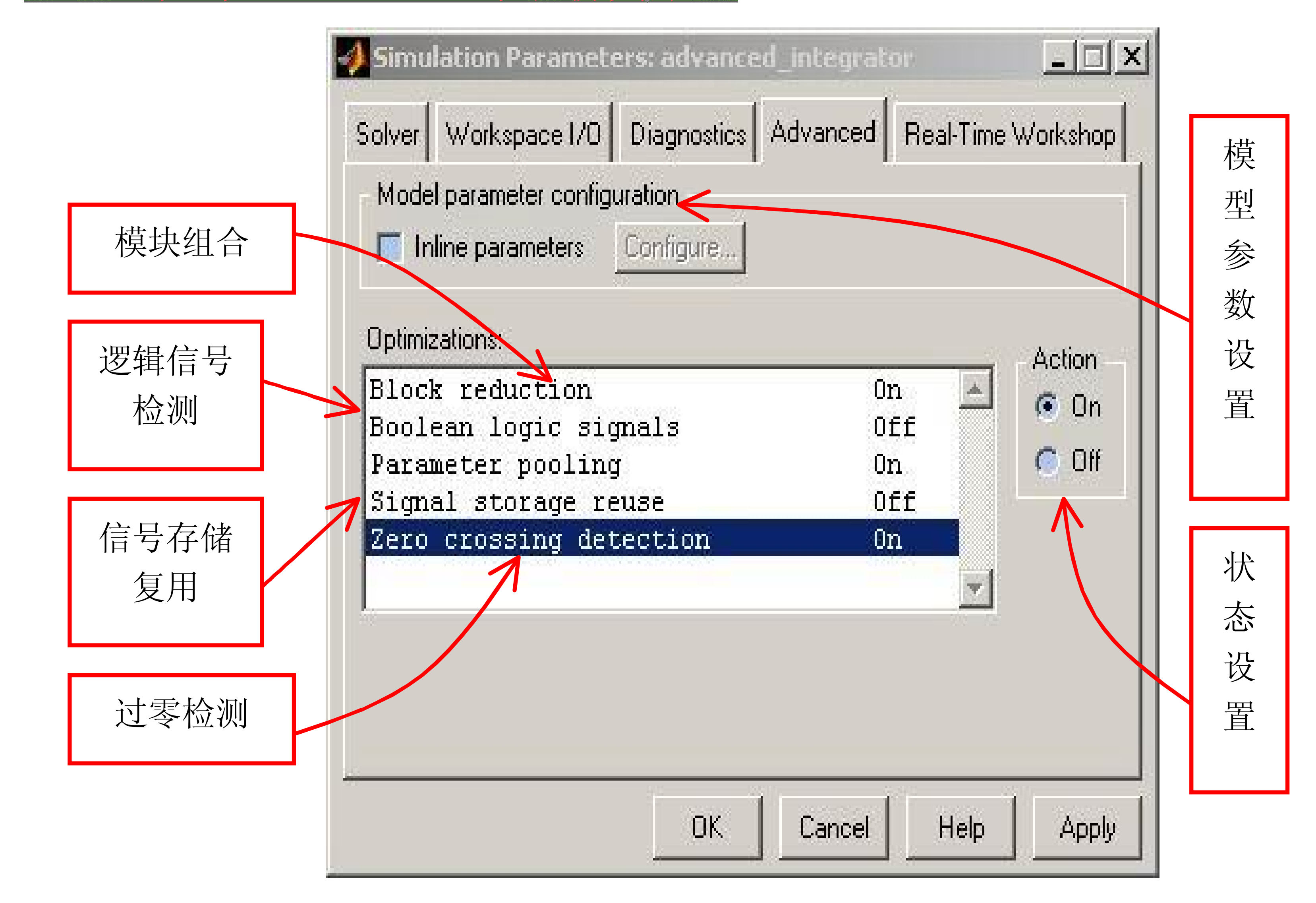


图6.19 仿真参数的高级选项卡

其中各个选项的意义如下：

(1) 状态设置（Action）：显示当前参数所处的状态，其取值为开（on）或关（off）。

(2) 模块组合（Block reduction）：使用一个合成模块替代一组模块。

(3) 逻辑信号检测（Boolean logic signals）：检测逻辑模块的输入是否为0。激活此状态（Action设置为on）可以强制模块输入为逻辑值。

(4) 信号存储复用（Signal storage reuse）：复用内存空间以节省信号所使用的内存。

(　　(5) 过零检测（Zero crossing detection）：检测过零事件的发生。

(6) 模型参数设置（Model parameter configuration）：设置整个系统模型的参数，此参数仅影响到在执行过程中可以改变系统参数的仿真或模型本身的参数可以由其它模型访问的系统模型产生的代码。

6.5.2 诊断选项

在对动态系统进行仿真时，很难避免一些问题的出现，如系统模型中存在的代数环结构、系统中数据连续的转换以及信号的越界等等。使用Simulink仿真参数对话框中的诊断选项可以对动态系统仿真过程中出现的问题进行诊断，同时也可以在系统仿真之前进行特定的测试，以有效地提高系统仿真的性能。

Simulation Parameters: advanced_integrator

Solver | Workspace I/O | Diagnostics | Advanced | Real-Time Workshop

Simulation options

Consistency checking: none　　Bounds checking: none

Configuration options:

Option	Setting
-1 sample time in source	Warning
Algebraic loop	Warning
Block priority violation	Warning
Check for singular matrix	None
Data overflow	Warning
int32 to float conversion	Warning
Invalid FcnCall connection	Error
Min step size violation	Warning
MultiTask rate transition	Error
S-function upgrades needed	None

Action

- None
- Warning
- Error

OK　Cancel　Help　Apply

一致性检测

边界检测

问题解决方法

仿真中可能出现的问题

图6.20　仿真参数之诊断选项卡



第7章　技Simulink子系统术



7.1 Simulink简单子系统概念：回顾与复习

7.1.1 通用子系统的生成

在使用Simulink子系统技术时，通常子系统的生成有如下两种方法：

(1) 在已经建立好的系统模型之中建立子系统（如图7.1所示）。首先选择能够完成一定功能的一组模块，然后选择Simulink模型创建编辑器中Edit菜单下的Create Subsystem，即可建立子系统并将这些模块封装到此子系统中，Simulink自动生成子系统的输入与输出端口。

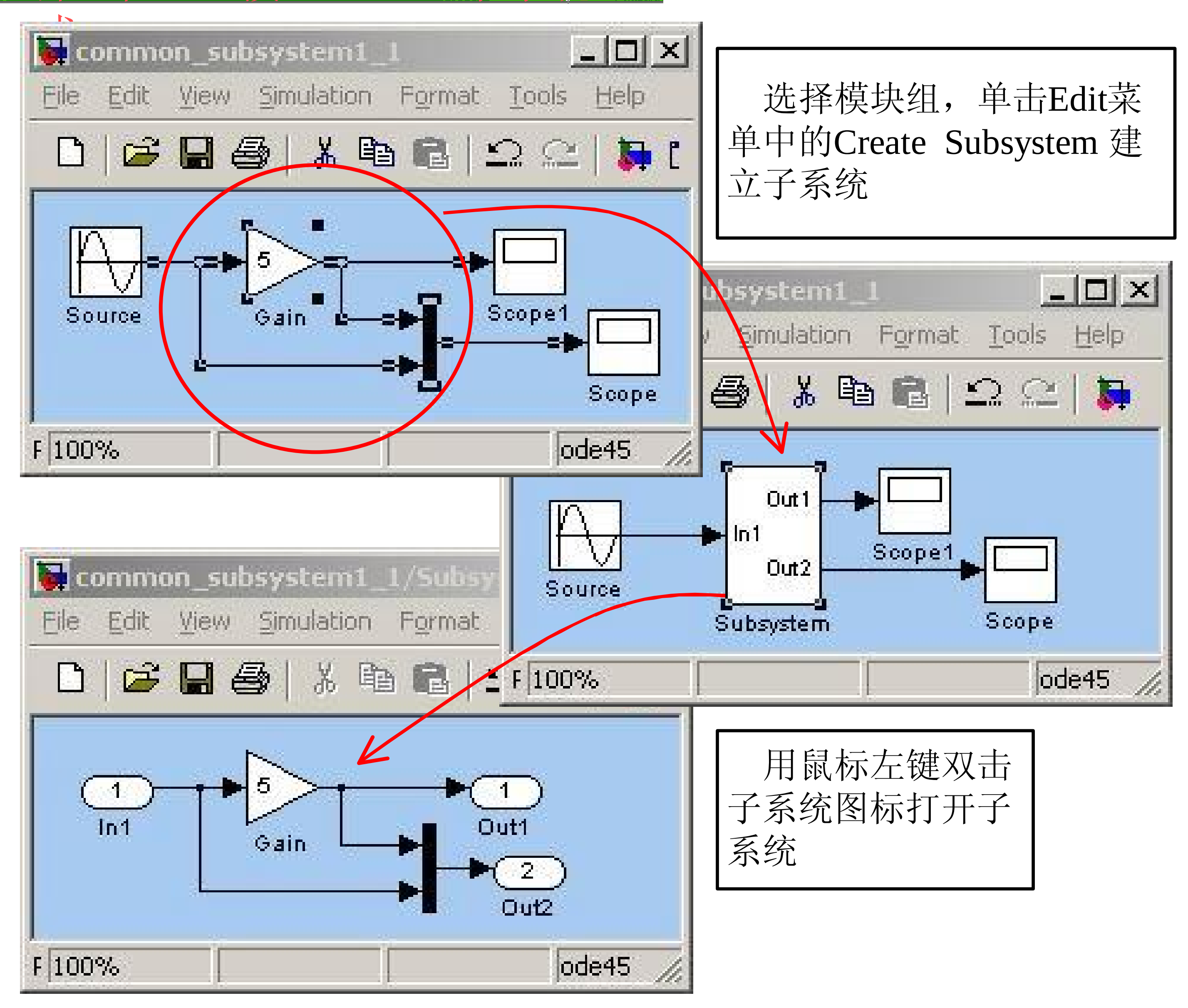


图7.1 子系统建立：由模块组生成子系统

(2) 在建立系统模型时建立空的子系统（如图7.2所示）。使用Subsystems模块库中的Subsystem模块建立子系统，首先构成系统的整体模型，然后编辑空的子系统内的模块。注意，对于多输入与多输出子系统而言，需要使用Sources模块库中的In1输入虚模块与Sinks模块库中的Out1输出虚模块来实现。

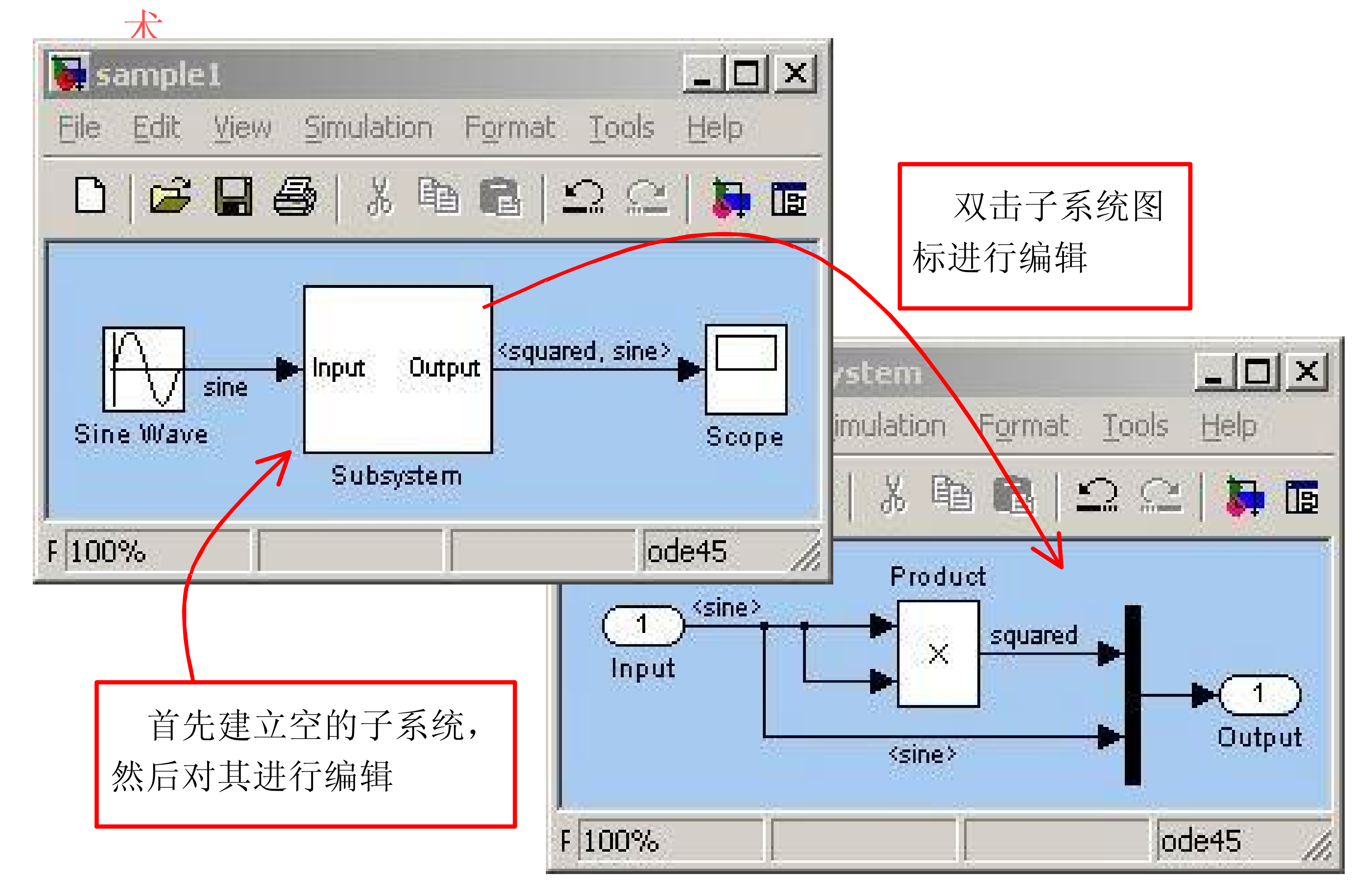


图7.2 子系统建立：生成空子系统并编辑

7.1.2子系统的基本操作

在用户使用Simulink子系统建立系统模型时，有几个简单的操作比较常用，这里做一个简单的列举：

(1) 子系统命名：命名方法与模块命名类似。为增强系统模型的可读性，应使用有代表意义的文字给子系统进行命名。

(2) 子系统编辑：用鼠标左键双击子系统模块图标，打开子系统以对其进行编辑。

(3) 子系统的输入：使用Sources模块库中的Inport输入模块（即In1模块）作为子系统的输入端口。

(4) 子系统的输出：使用Sinks模块库中的Outport输出模块（即Out1模块）作为子系统的输出端口。



7.2　Simulink高级子系统技术

条件执行子系统的执行受到控制信号的控制，根据控制信号对条件子系统执行的控制方式的不同，可以将条件执行子系统划分为如下的几种基本类型。

(1) 使能子系统：是指当控制信号的值为正时，子系统开始执行。

(2) 触发子系统：是指当控制信号的符号发生改变时（也就是控制信号发生过零时），子系统开始执行。触发子系统的触发执行有三种形式：

① 控制信号上升沿触发：控制信号具有上升沿形式。

② 控制信号下降沿触发：控制信号具有下降沿形式。

③ 控制信号的双边沿触发：控制信号在上升沿或下降沿时触发子系统。

(3) 函数调用子系统：这时条件子系统是在用户自定义的S-函数中发出函数调用时开始执行。有关S-函数的概念将在后续章节中介绍。

7.2.1 条件执行子系统的建立方法

在进一步介绍条件执行子系统之前，首先介绍如何建立条件执行子系统（如图7.3所示）。其中需要使用Subsystems模块库中的Enabled Subsystem（使能子系统）模块、TriggeredSubsystem（触发子系统）模块及Enabled and Triggered Subsystem（使能触发子系统）模块。

在建立条件执行子系统前需要注意以下两点：

(1) 对于Simulink的早期版本而言，不存在专门的Subsystems模块库。

(2)　Simulink系统模型的最高层不允许使用Enable与Trigger信号，而仅允许在子系统中使用。

图7.3中并没有建立一个完整的动态系统的模型，而仅仅是给出建立条件执行子系统的方法，因此并没有给出执行系统所需的使能信号源与触发信号源（如图7.3中椭圆曲线所示，使能输入端与触发输入端采用不同的信号标志）。如果此时用户运行此系统进行仿真，MATLAB命令窗口中会给出输入端口没有信号连接的警告，而且系统的输出均为0。

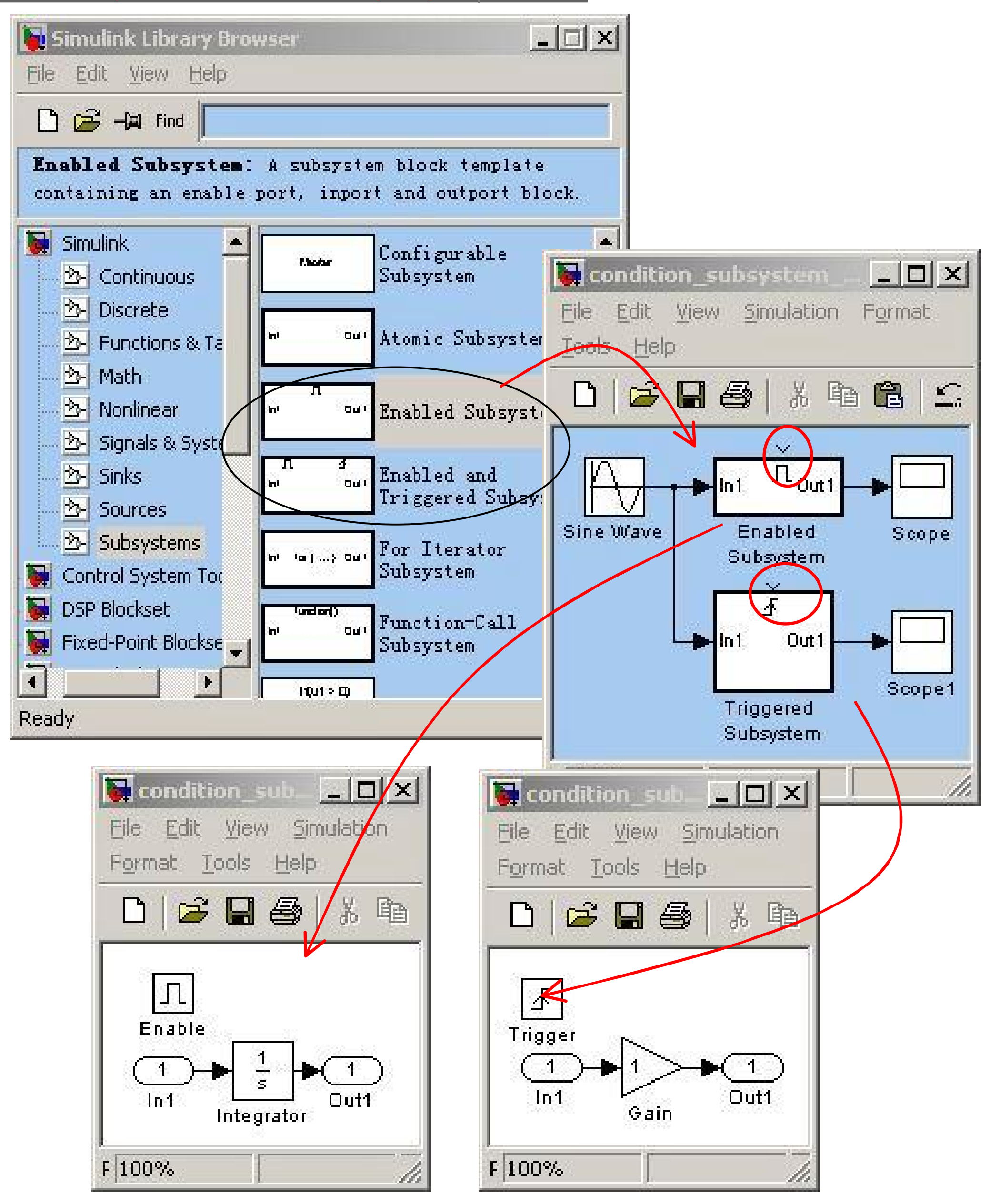


图7.3 条件执行子系统的建立方法示意图

7.2.2 使能子系统

在对具体的条件执行子系统进行介绍时，采用实际的系统模型进行说明非常有利于读者的理解。因此，本书均以实际的系统模型为例进行说明。

所谓的使能子系统，是指只有当子系统的使能信号输入为正时，子系统才开始执行。

【例7.1】 使能子系统的建立与仿真。

按照7.1节中的方法建立如图7.4所示的动态系统模型。

在此系统模型中，存在着两个由方波信号驱动的使能子系统（图中虚线框所框的子系统，以A与B表示）。当控制信号（即系统模型中的方波信号）为正时开始执行子系统A，控制信号为负时（方波信号经过一反相信号操作，由Math模块库中的Logical Operator逻辑操作模块NOT操作符实现）开始执行子系统B。图7.5所示为使能子系统A与B的结构以及相应的使能状态设置。

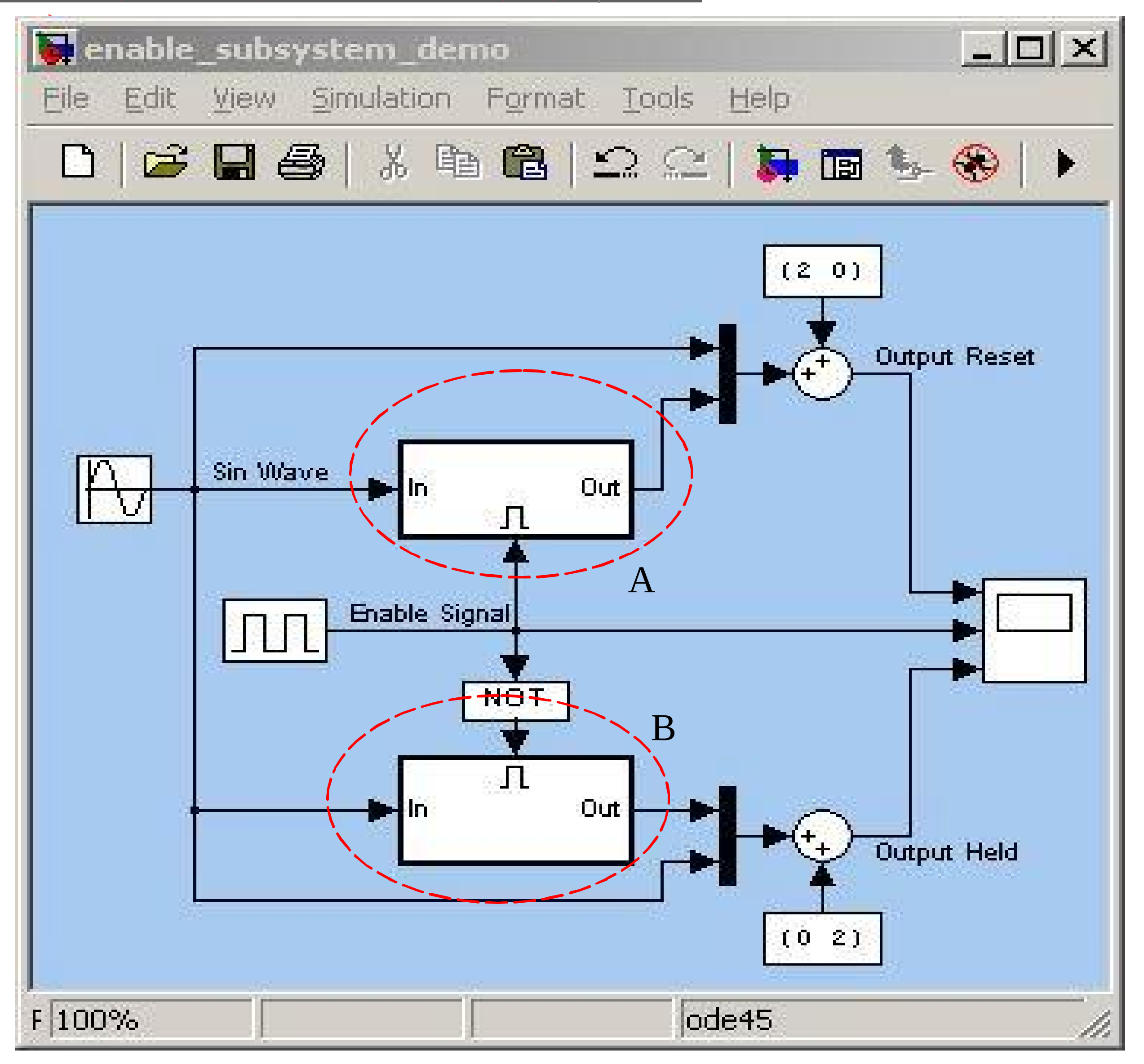

enable_subsystem_demo
File Edit View Simulation Format Tools Help
Sin Wave
In
Out
Enable Signal
NOT
A
B
[2 0]
[0 2]
Output Reset
Output Held
100%
ode45

图7.4 使能子系统模型

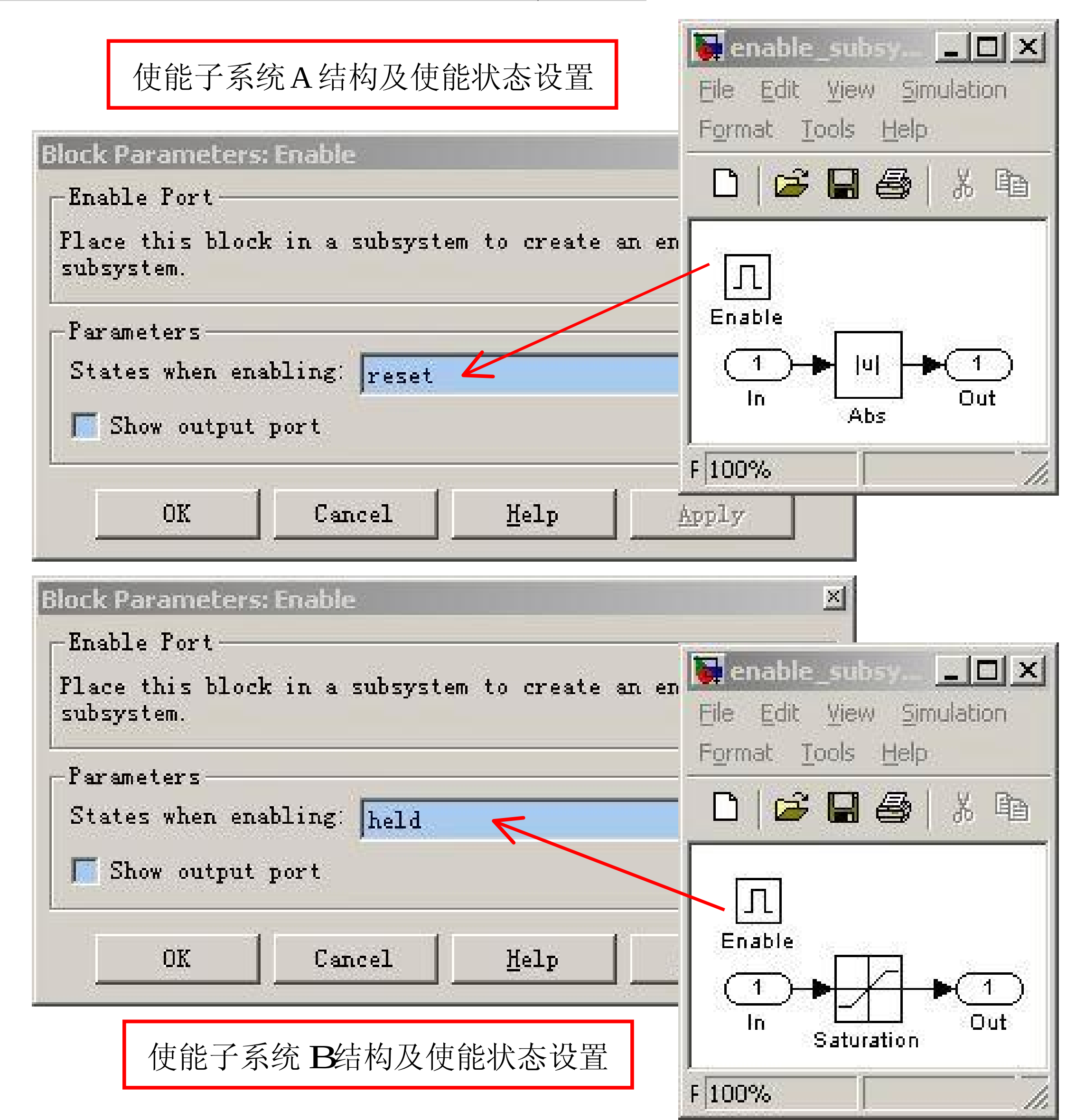


图7.5 使能子系统A、B的内部实现及使能参数设置

此系统模型中各模块的参数设置如下：

(1) 系统输入为采用默认设置的正弦信号（即单位幅值，单位频率的单位正弦信号）。

(2) 使能子系统的控制信号源，使用Sources模块库中的Pulse Generator脉冲信号发生器所产生的方波信号。其设置为：脉冲周期（Period）为5 s，其余采用默认设置。

(3) 使能子系统A中的使能信号，其使能状态设置为重置reset；使能子系统B中的使能信号，其使能状态设置为保持held。

(4) 下方使能子系统中饱和模块（Saturation），其参数设置为：饱和上限为0.75，饱和下限为-0.75。

(5) 偏移常数信号，其参数设置分别为[2　0]与[0 2]，如图7.4中系统模型所示。

(6) 系统输出Scope模块参数设置，如图7.6所示。

系统仿真参数设置如下：

(1) 仿真时间：设置仿真时间范围为0至20 s。

(2) 求解器设置：采用默认设置，即连续变步长，具有过零检测能力的求解器。

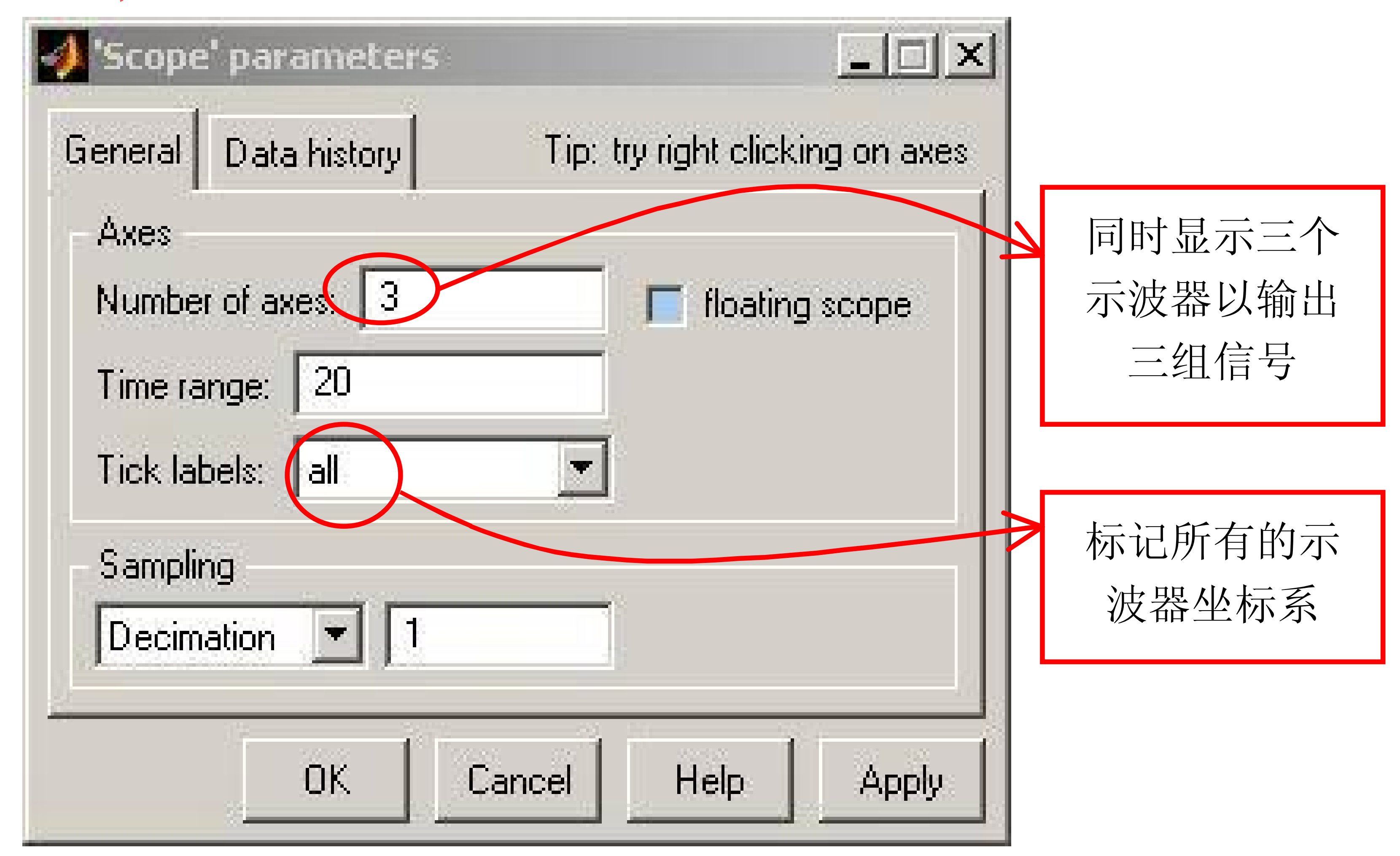


图7.6 Scope模块参数设置

从图7.7中可以明显看出，只有在控制信号为正时，使能子系统才输出，而且设置不同的使能状态可以获得不同的结果（结果被重置或被保持）。对于采用状态重置的使能子系统A，其输出被重置；而采用状态保持的使能子系统B，其输出被保持。如果在使能子系统中存在着状态变量，那么当使能模块状态设置为重置时，它的状态变量将被重置为初始状态（可能不为零），当使能模块设置为保持时，它的状态变量将保持不变。至于系统的输出，取决于系统的状态变量以及系统的输入信号，这里不再赘述。

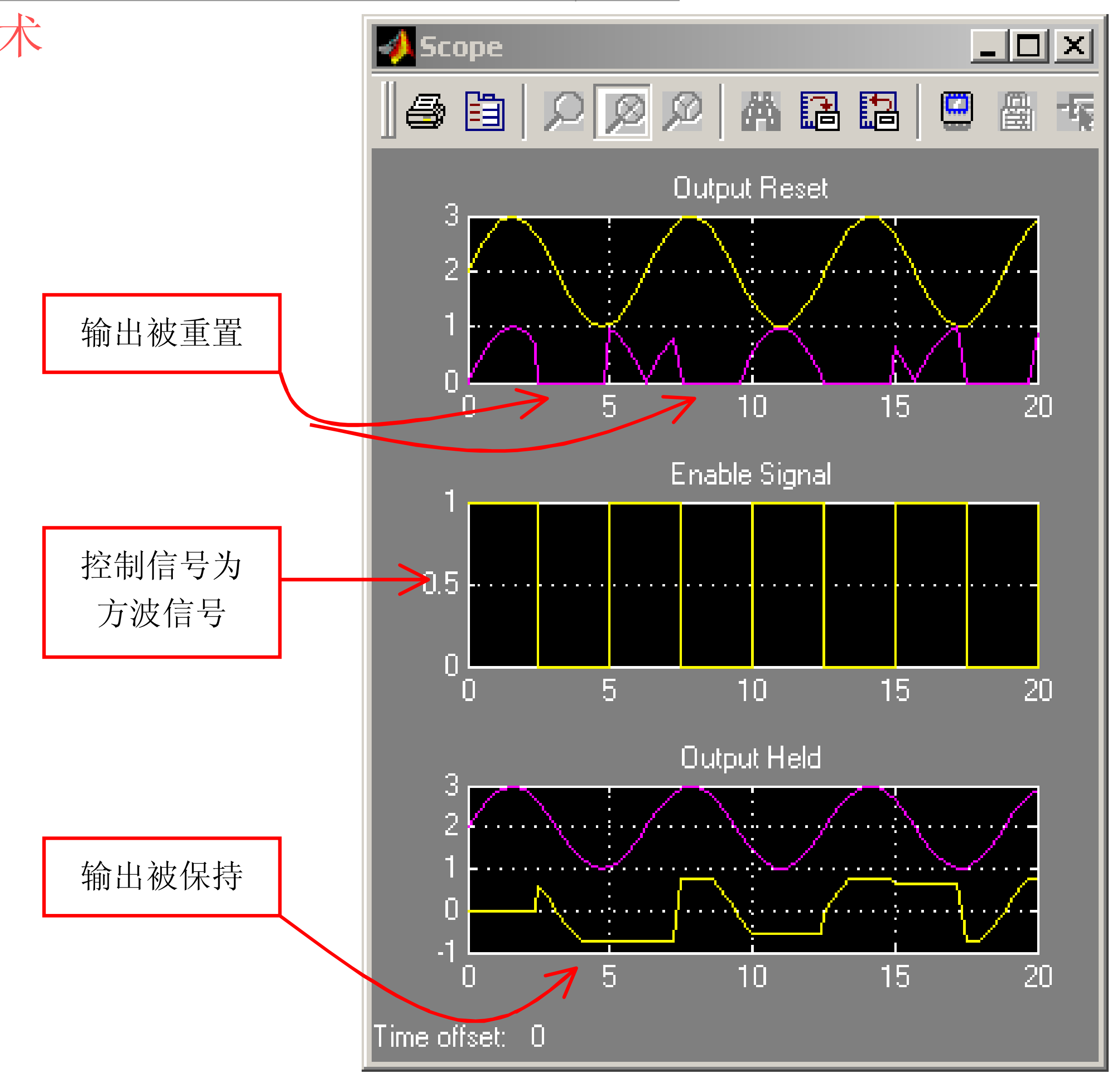


图7.7 【例7.1】中使能子系统的仿真结果

7.2.3 触发子系统

所谓的触发子系统指的是只有在控制信号的符号发生改变的情况下（也就是控制信号出现过零事件时），子系统才开始执行。如前所述，根据控制信号符号改变方式的不同可以

(1) 上升沿触发子系统。系统在控制信号出现上升沿时开始执行。

(2) 下降沿触发子系统。系统在控制信号出现下降沿时开始执行。

(3) 双边沿（上升沿或下降沿）触发子系统。系统在控制信号出现任何过零时开始执行。

【例7.2】 触发子系统的建立与仿真分析。

在这个例子中，存在着三个使用不同触发方式的触发子系统，分别是上升沿触发、下降沿触发以及双边沿触发。图7.8所示为此系统的系统模型。

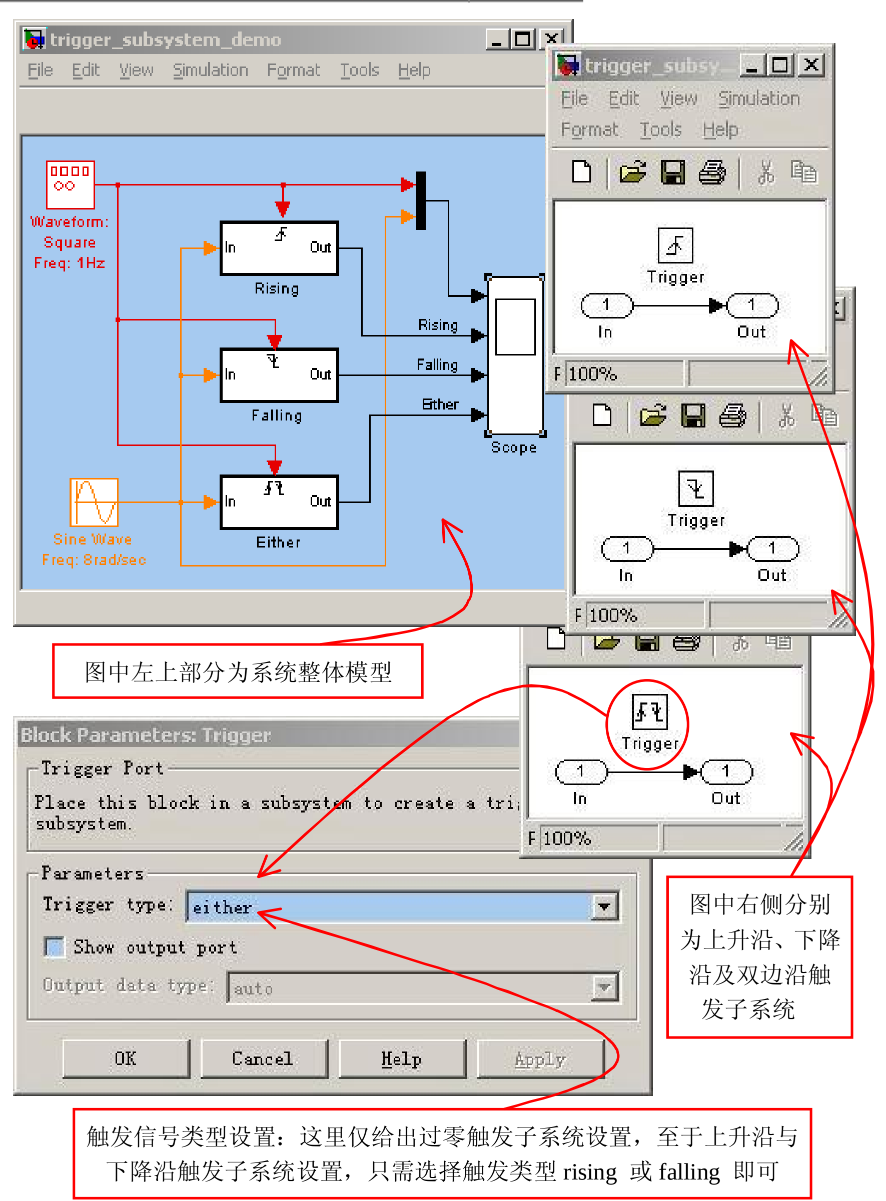


图7.8 触发子系统的系统模型与触发类型设置

下面给出系统模型中各个系统模块的参数以及仿真参数设置，最后给出系统仿真结果并进行一定的分析。模块参数设置如下：

(1) 系统输入正弦信号，其模块参数除频率选择为8 rad/sec外，其余采用默认的参数。

(2) 系统触发控制信号为方波信号，使用Sources模块库中的信号发生器Signal Generator模块生成方波信号，其参数设置为：波形waveform为方波square，幅值为0.5，频率为1 Hz。

(3) 系统输出Scope模块。在Scope模块参数设置中，设置其坐标轴数目为4，以使用4个示波器同时显示4组信号。其方法与使能子系统中一样，这里不再赘述。

(4) 各触发子系统参数设置如图7.8中所示。分别设置其触发方式为Rising、Falling及Either即可。

系统仿真参数设置如下：

(1) 仿真时间范围0至8 s。

(2) 采用变步长连续求解器，最大步长为0.01，以避免信号的不连续。

运行此系统进行仿真，仿真结果如图7.9所示。其中第一个示波器输出为系统输入正弦

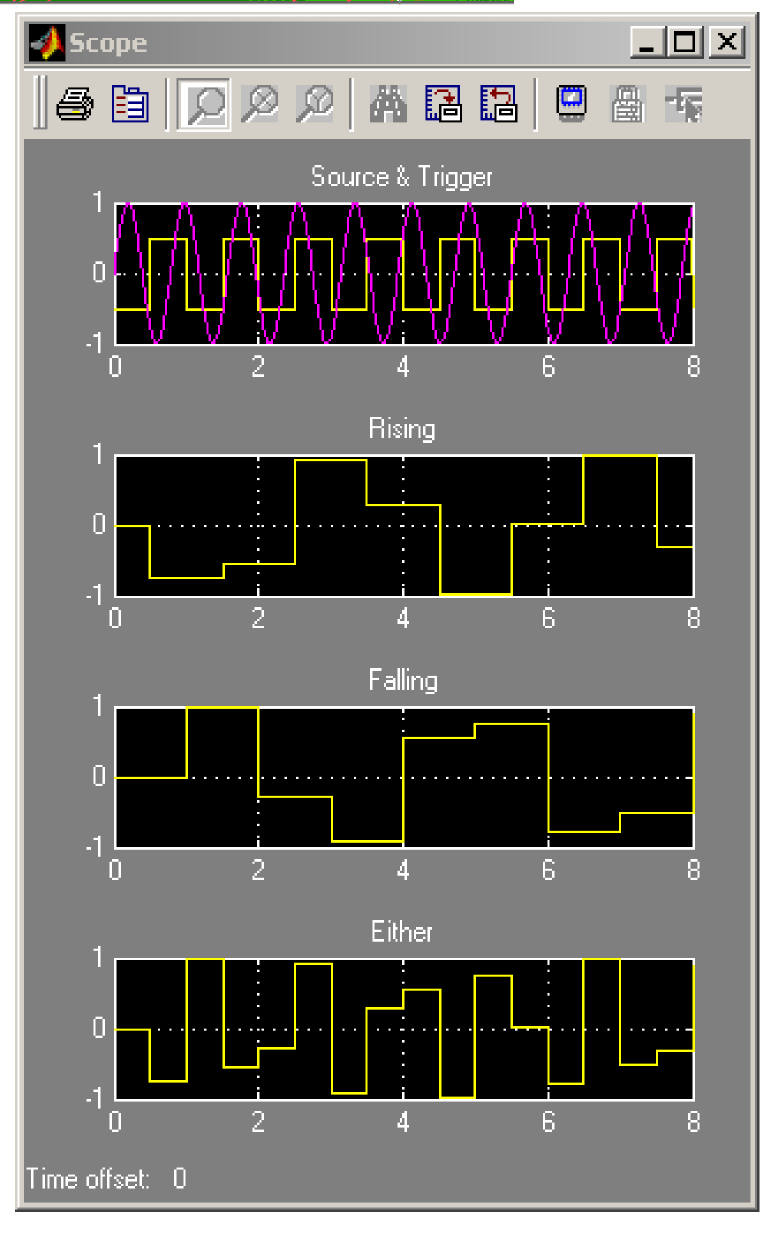


图7.9 触发子系统的仿真结果

7.2.4 触发使能子系统

在介绍条件执行子系统时已经提到，对于某些条件执行子系统而言，其控制信号可能不止一个。在很多情况下，条件执行子系统同时具有触发控制信号与使能控制信号，这样的条件执行子系统一般称之为触发使能子系统。顾名思义，触发使能子系统指的是子系统的执行受到触发信号与使能信号的共同控制，也就是说，只有当触发条件与使能条件均满足的情况下，子系统才开始执行。触发使能子系统的工作原理如图7.10所示。

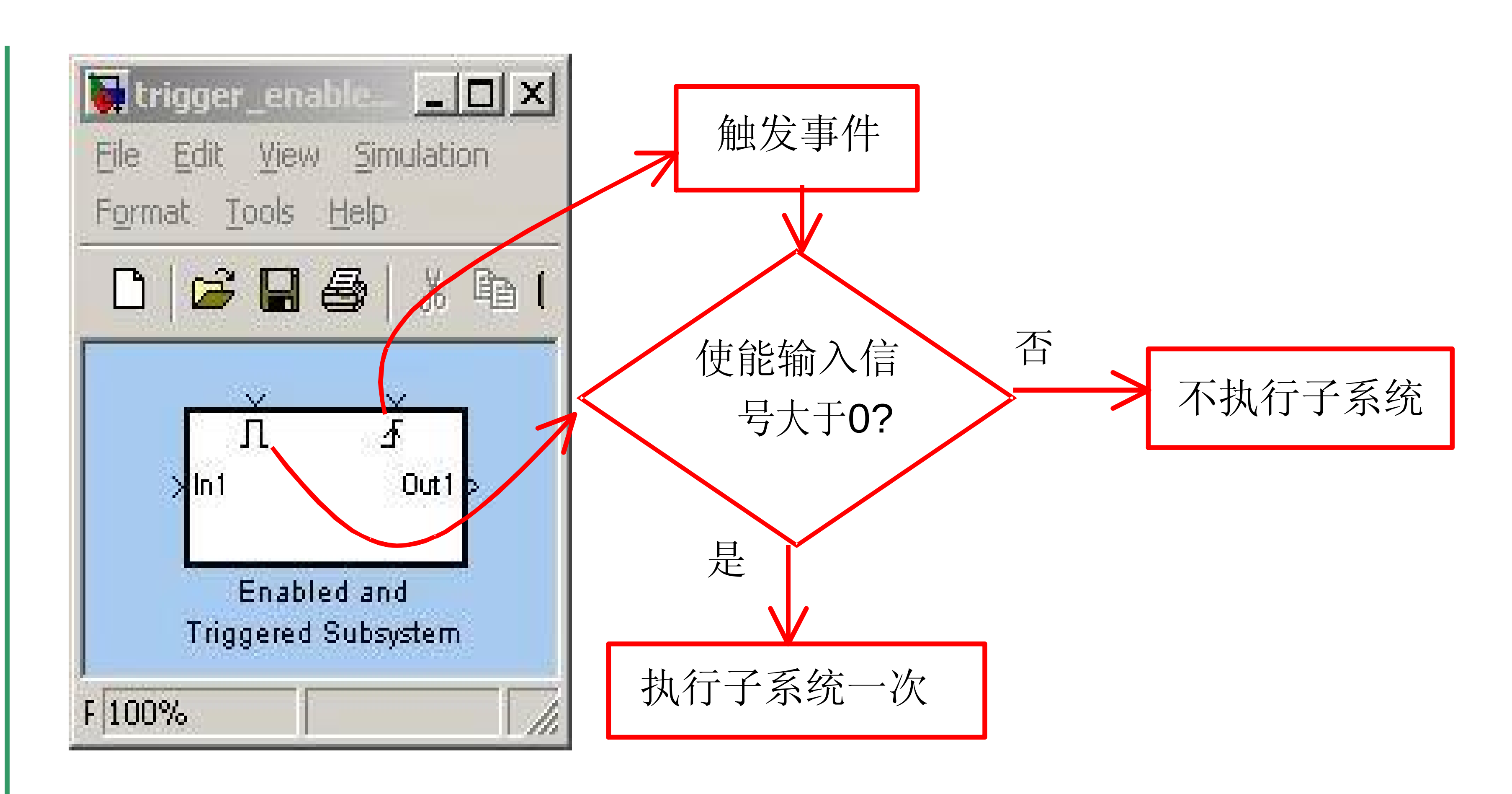


图7.10 触发使能子系统的工作原理

7.2.5 原子子系统

1. 原子子系统的概念

至此，本章已经介绍了通用子系统、使能子系统以及触发子系统等三种不同子系统的概念及其使用。虽然子系统都可以将系统中相关模块组合封装成一个单独的模块，大大方便了用户对复杂系统的建模、仿真及分析；但是对于不同的子系统而言，它们除了有如上的共同点之外，还存在着本质上的不同。下面介绍此三种子系统的不同本质，并简单介绍原子子系统的概念。

虚子系统具有如下的特点：

(1) 子系统只是系统模型中某些模块组的图形表示。

(2) 子系统中的模块在执行时与其上一级模块统一被排序，不受子系统的限制。

(3) 在一个仿真时间步长内，Simulink可以多次进出一个子系统。

原子子系统具有如下的特点：

(1) 子系统为一“实际”的模块，需要按照顺序连续执行。

(2) 子系统作为一个整体进行仿真，其功能类似于一个单独的系统模块。

(3) 子系统中的模块在子系统中被排序执行。

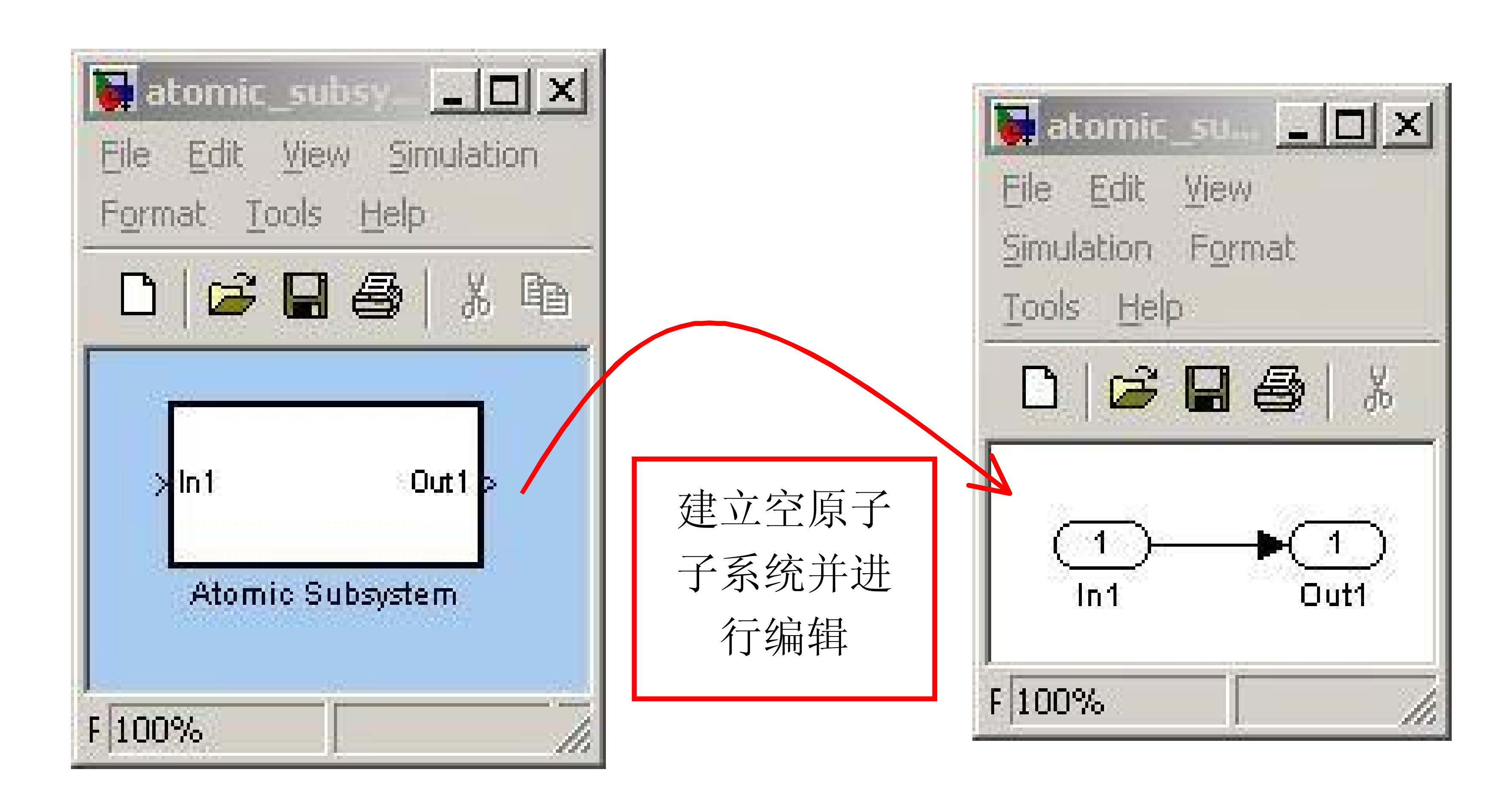
2. 建立原子子系统

触发子系统在触发事件发生时，子系统中的所有模块一同被执行，当所有的模块都执行完毕后，Simulink才开始执行上一级其它模块或其它子系统，因此触发子系统为一原子子系统。

在Simulink中有两种方法可以建立原子子系统：

(1) 建立一个空的原子子系统。

(2) 将已经建立好的子系统强制转换为原子子系统。


atomic_subsy...
File Edit View Simulation
Format Tools Help
In1
Out1
Atomic Subsystem
100%
建立空原子子系统并进行编辑
atomic_su...
File Edit View
Simulation Format
Tools Help
1
In1
1
Out1
100%

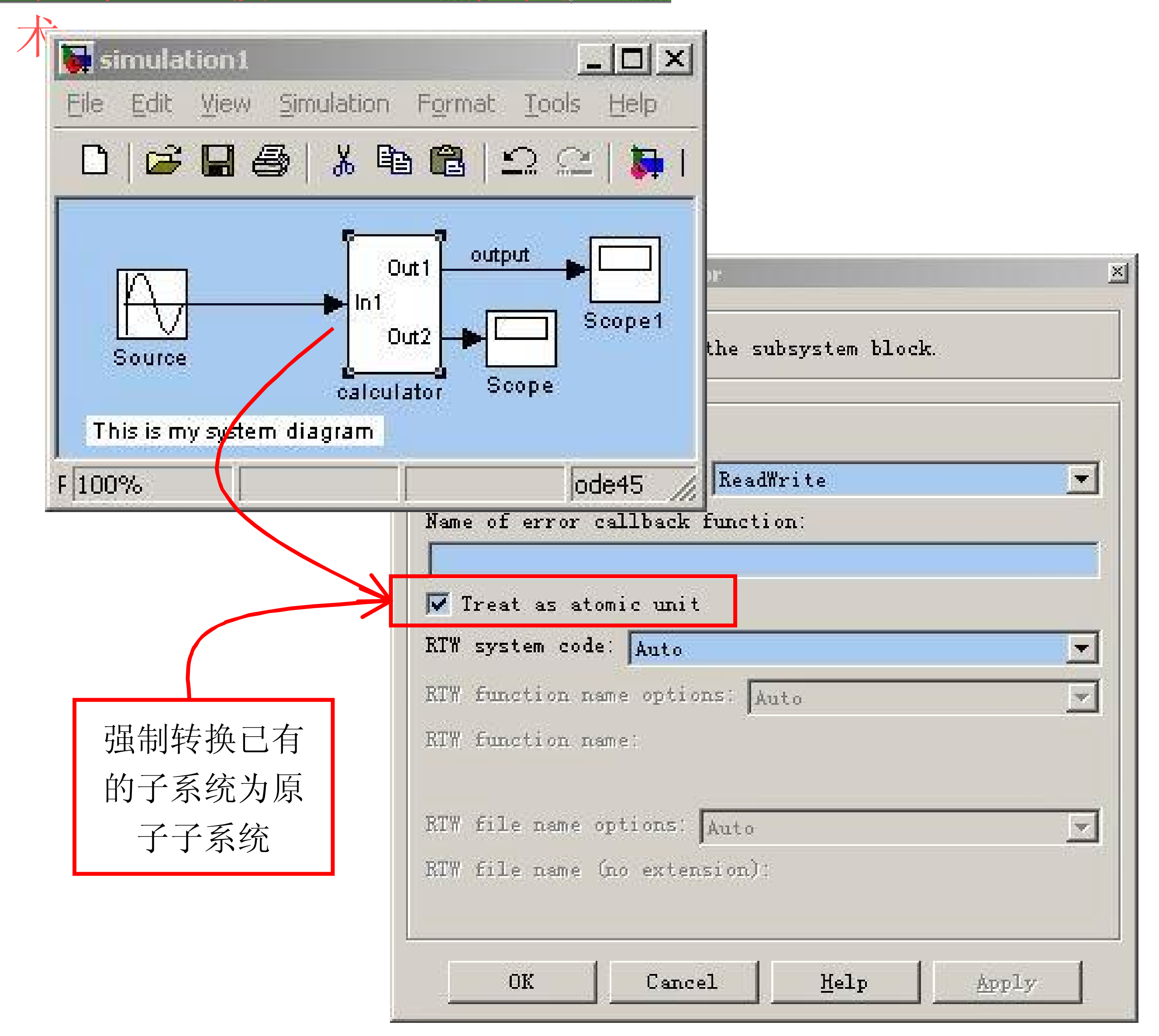


图7.11 原子子系统的建立方法

7.2.6 其它子系统介绍

在Simulink Block Library（Simulink模块库，版本4.1）中的Subsystems子系统模块库中除了前面所介绍的通用子系统、触发子系统、使能子系统以及原子子系统之外，Simulink还提供了许多其它的条件执行子系统。图7.12所示为Subsystems模块库中的所有子系统模块。在此对其进行简单的介绍。

(1) 可配置子系统（Configurable Subsystem）：用来代表用户自定义库中的任意模块，只能在用户自定义库中使用。

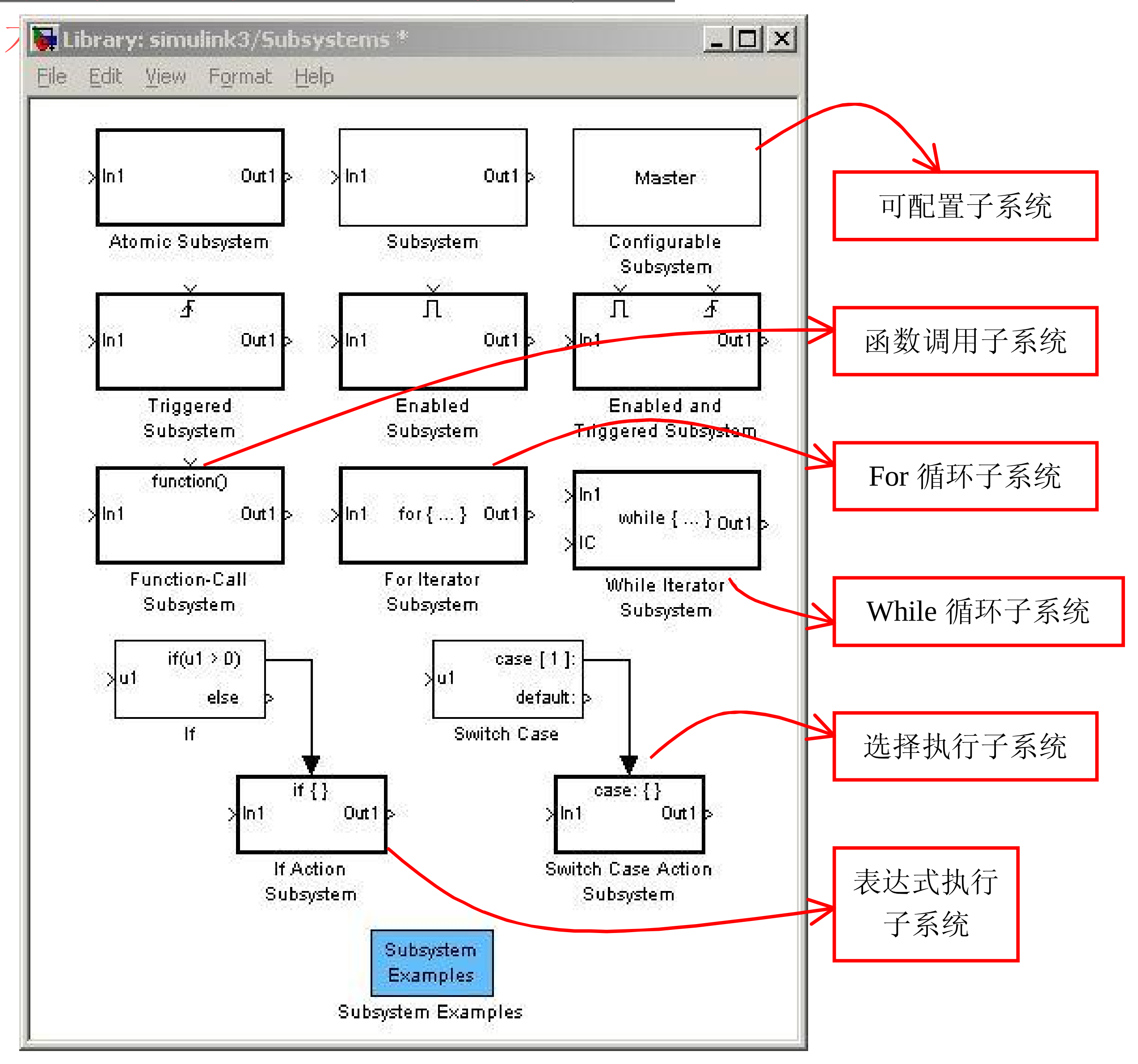


图7.12 Subsystems模块库中的所有子系统模块

(2) 函数调用子系统（Function-Call Subsystem）：使用S-函数的逻辑状态而非普通的信号作为触发子系统的控制信号。函数调用子系统属于触发子系统，在触发子系统中触发模块Trigger的参数设置中选择Function-Call可以将由普通信号触发的触发子系统转换为函数调用子系统，如图7.13所示。

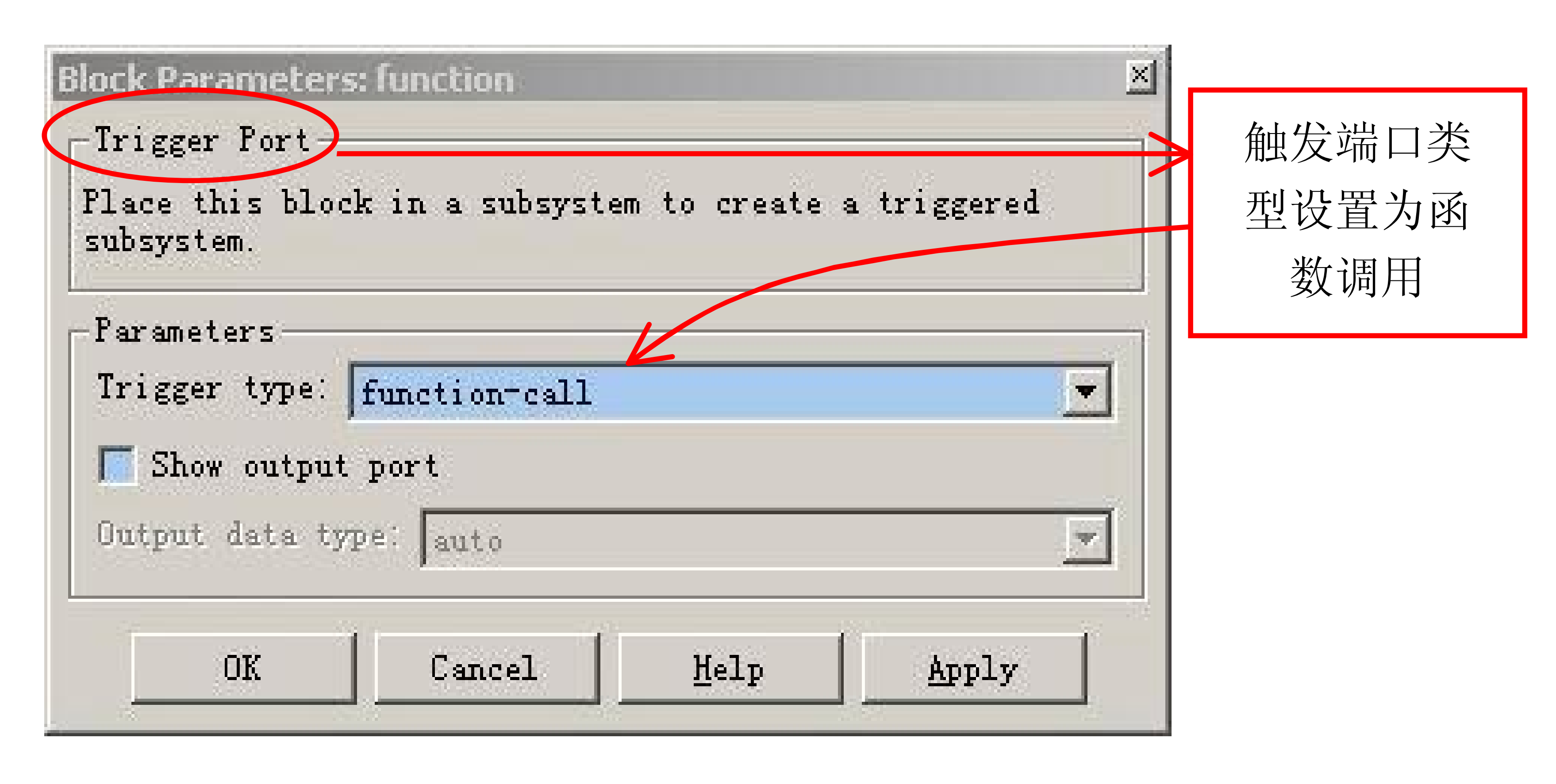


图7.13 函数调用触发类型设置

(3) For循环子系统（For Iterator Subsystem）：For循环子系统的目的是在一个仿真时间步长之内循环执行子系统。

(4) While循环子系统（While Iterator Subsystem）：与For循环子系统相类似，While循环子系统同样可以在一个仿真时间步长内循环执行子系统，但是其执行必须满足一定的条件。

(5) 选择执行子系统（Switch Case Action Subsystem）：在某些情况下，系统对于输入的不同取值，分别执行不同的功能。

(6) 表达式执行子系统（If Action Subsystem）：为了与前面的条件执行子系统相区别，这里我们称If Action Subsystem为表达式执行子系统。此子系统的执行依赖于逻辑表达式的取值，这与C语言中的If Else语句类似。需要注意的是，表达式执行子系统必须同时使用If模块与If Action Subsystem模块（均在Subsystems模块库中）。



7.3 Simulink的子系统封装技术

7.3.1如何封装子系统

封装子系统与建立子系统并不相同，建立子系统指的是将具有一定功能的一组模块“容纳”在一个子系统之中，使用单一图形方式的子系统模块来表示一组模块，从而增强系统模型的可读性，在动态系统进行仿真时需要对子系统中各个模块的参数分别进行设置；而封装子系统指的是将已经建立好的具有一定功能的子系统进行封装，封装的目的在于生成用户自定义的模块，此模块与子系统的功能完全一致。

封装子系统具有如下特点：

(1) 自定义子系统模块及其图标。

(2) 用户双击封装后的图标时显示子系统参数设置对话框。

(3) 用户自定义子系统模块的帮助文档。

(4) 封装后的子系统模块拥有自己的工作区。

因此，使用封装子系统技术具有以下优点：

(1) 向子系统模块中传递参数，屏蔽用户不需要看到的细节。

(2) “隐藏”子系统模块中不需要过多展现的内容。

(3) 保护子系统模块中的内容，防止模块实现被随意篡改。

【例7.3】　以第5章中人口动态变化的非线性离散模型为例说明子系统的封装技术。

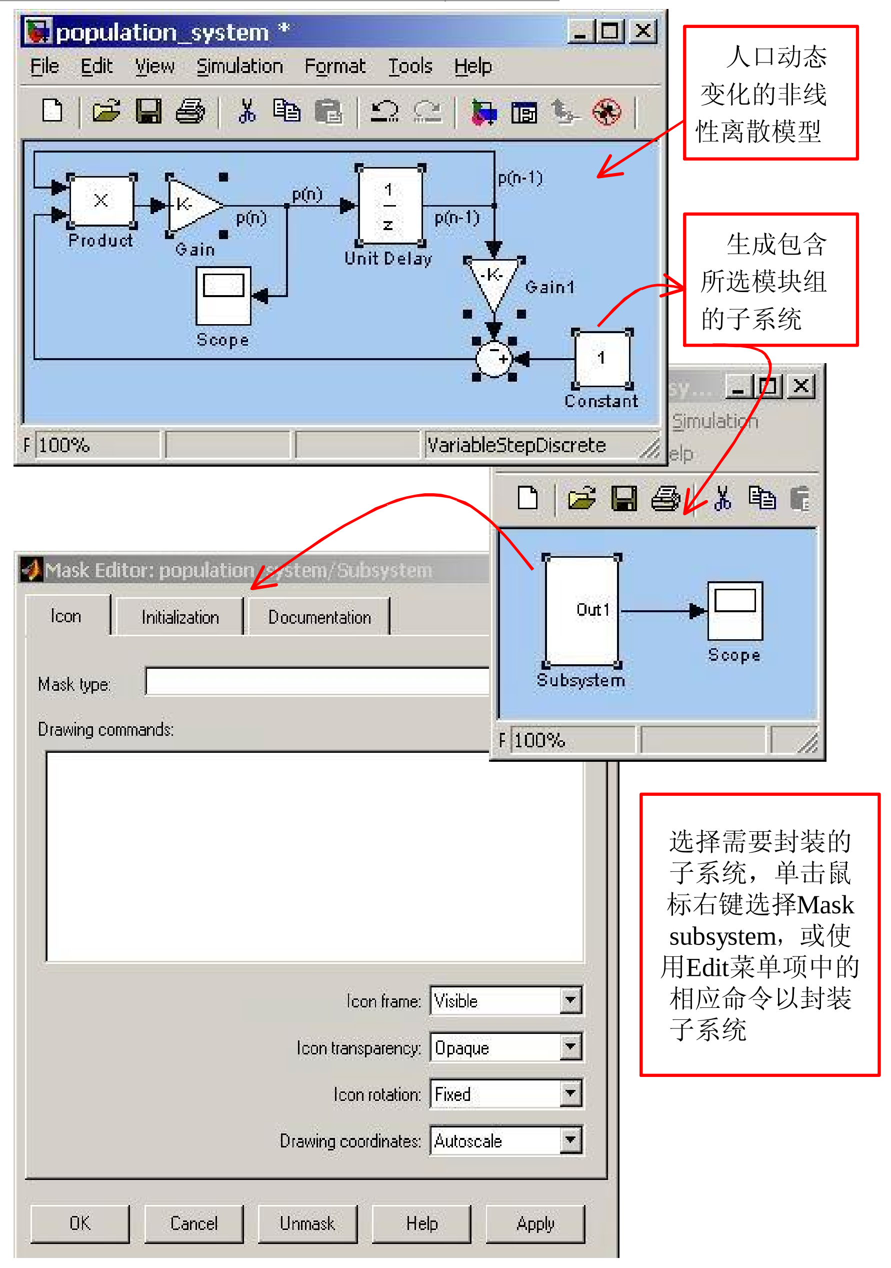
解：封装子系统的基本过程如下：

(1) 打开人口动态变化的非线性离散模型框图。

(2) 生成需要进行封装的子系统。

(3) 选择需要封装的子系统，单击鼠标右键选择Mask subsystem，或使用Edit菜单项中的相应命令进行子系统封装。

封装子系统的基本流程图如图7.14所示，图中上方为系统原始模型框图，中间为使用子系统的系统模型框图。



人口动态变化的非线性离散模型

生成包含所选模块组的子系统

选择需要封装的子系统，单击鼠标右键选择Mask subsystem，或使用Edit菜单项中的相应命令以封装子系统

图7.14 子系统封装流程示意图

7.3.2 封装编辑器之图标编辑对话框

当选择Mask subsystem菜单命令进行子系统封装时，将出现如图7.15所示的封装编辑器并显示图标编辑对话框。

使用此编辑器可以对封装后的子系统进行各种编辑。这里首先介绍图标编辑对话框的功能与使用。

在默认情况下，封装子系统不使用图标。但友好的子系统图标可使子系统的功能一目了然。为了增强封装子系统的界面友好性，用户可以自定义子系统模块的图标。只需在图标编辑对话框中的子系统模块图标绘制命令栏（Drawing Commands）中使用MATLAB中相应的命令便可绘制模块图标，并可设置不同的参数控制图标界面的显示。下面逐一介绍各对话框的使用。

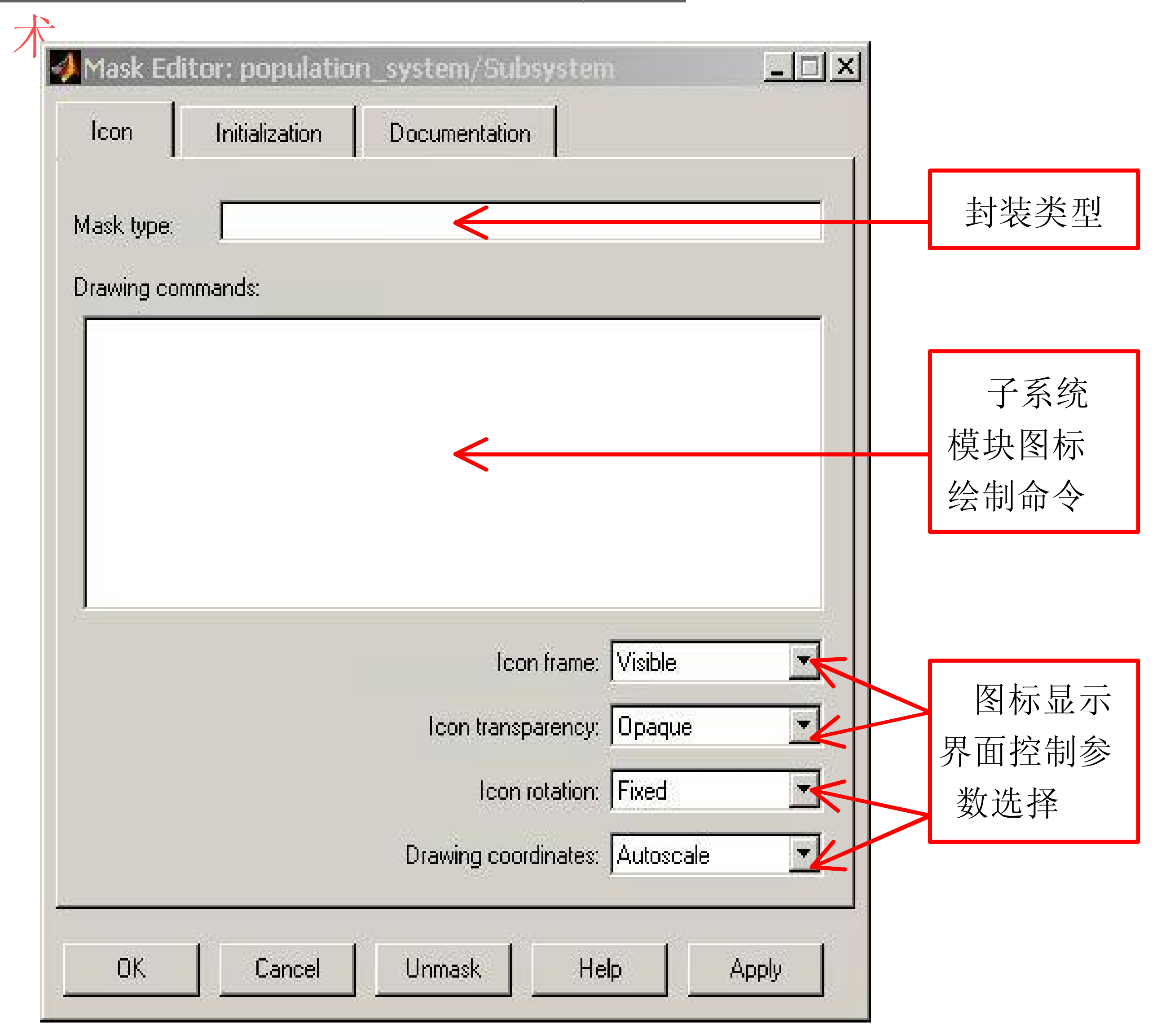


图7.15 封装编辑器之图标编辑对话框

1. 封装类型（Mask Type）

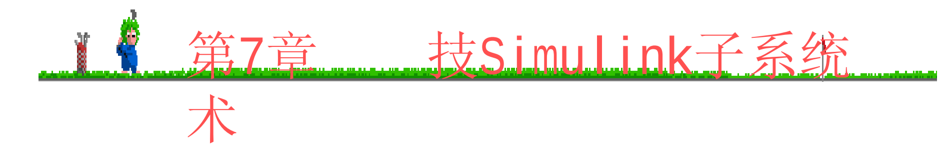
封装类型用来对封装后的子系统进行简短的说明。在此例中，用户可以键入Population Sample Mask。它将显示在参数对话框的左上角。

2. 图标显示界面控制参数

通过设置不同的参数可使模块图标具有不同的显示形式。控制参数共有四种。

1) 图标边框设置（Icon frame）

功能：设置图标边框为可见（Visible）或不可见（Invisible），如图7.16所示，其中左侧表示图标边框可见，而右侧表示边框不可见。



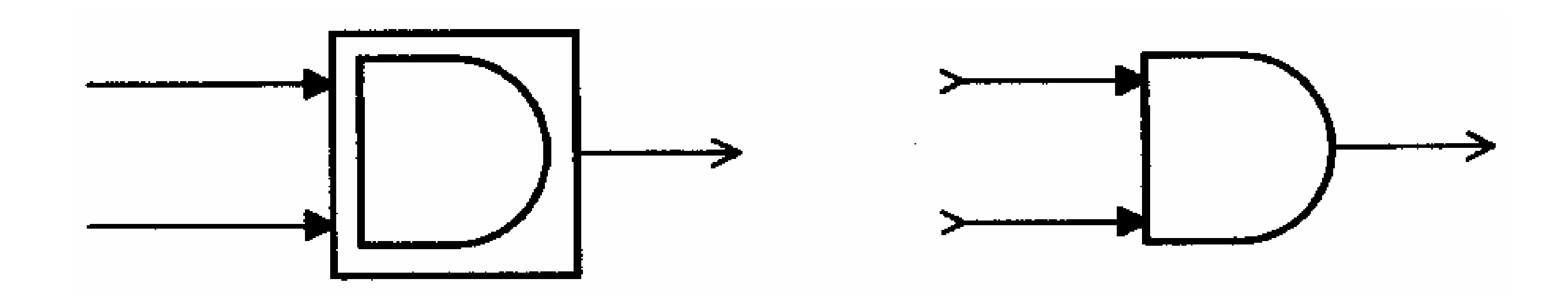


图7.16 边框设置示意图

2) 图标透明性设置（Icon transparency）

功能：设置图标为透明（Transparency）或不透明（Opaque）显示，如图7.17所示，其中左侧为图标不透明，而右侧表示图标透明（此时在图标后面的内容如模块端口标签可以被显示出）。

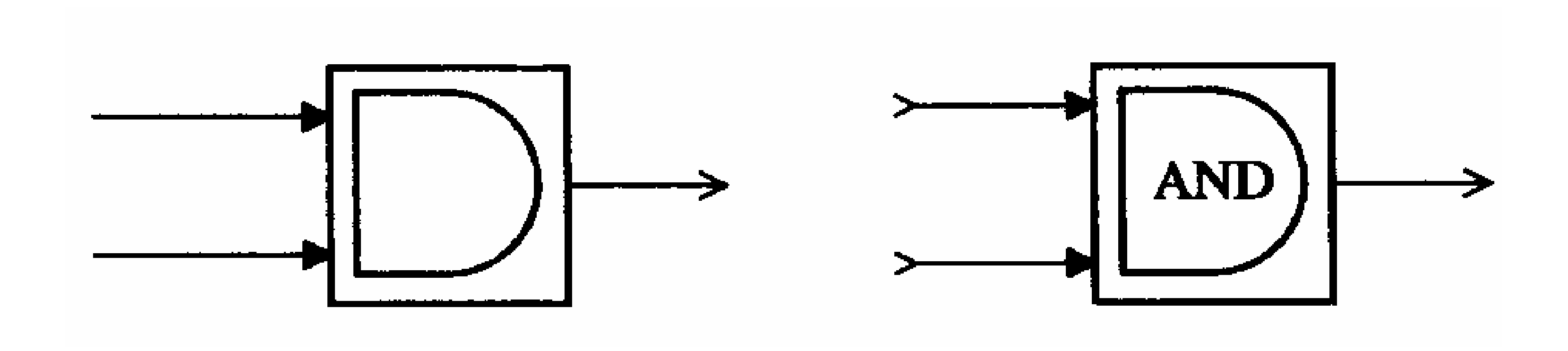


图7.17 图标透明显示设置示意

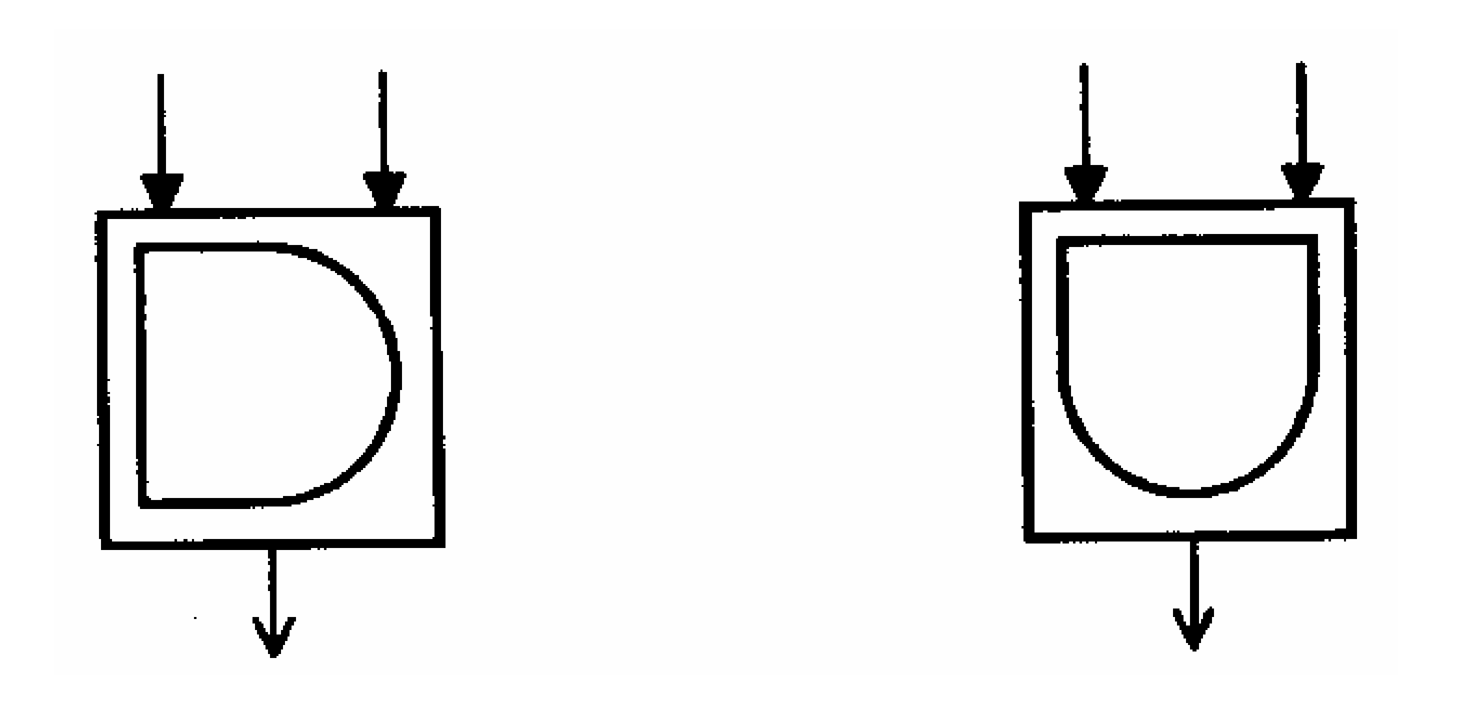


图7.18 图标旋转显示设置示意图

3) 图标旋转性设置（Icon rotation）

功能：设置图标为固定（Fixed）或可旋转（Rotates）显示，如图7.18所示，其中左侧表示图标不可旋转，而右侧表示图标可以旋转（图标随模块的旋转而旋转）。

4) 图标绘制坐标系设置（Drawing coordinates）

功能：设置图标绘制命令所使用的坐标系单位，仅对plot与text命令有效。其选项分别为自动缩放（Autoscale）、像素（Pixels）以及归一化表示（Normalized）。

3. 图标绘制命令栏（Drawing commands）

封装后子系统模块的图标均是在图标绘制命令栏中绘制完成的。使用不同的绘制命令可以生成不同的图标如描述性的文本、系统状态方程、图像以及图形等。如果在此栏中键入多个绘制命令，则图标的显示会按照绘制命令顺序显示。

1) 图标为描述性文本

使用如下的绘制命令可以在模块图标上显示文本：

```
disp('text')                                % text表示图标文本
disp(variablename)                          % variablename为工作空间中的字符串变量名
text(x, y, 'text')
text(x, y, stringvariablename)              % stringvariablename为已存在的字符串变量名
text(x, y, text, 'horizontalAlignment', halign, 'verticalAlignment', valign)
%halign与valign分别表示文本水平与垂直对齐方式，其取值不再赘述
fprintf('text')
fprintf('format', variablename)             % format表示文本的格式
```

port_label(port_type, port_number, label)　% 此命令可以显示模块的端口名称，其中port_type为端口类型，取值为'input　'或'output‘ % port_number为端口数目，label为端口文本

如果需要显示多行文本，可以使用\n表示换行。这时封装后的子系统图标为描述性文本，如图7.19所示。

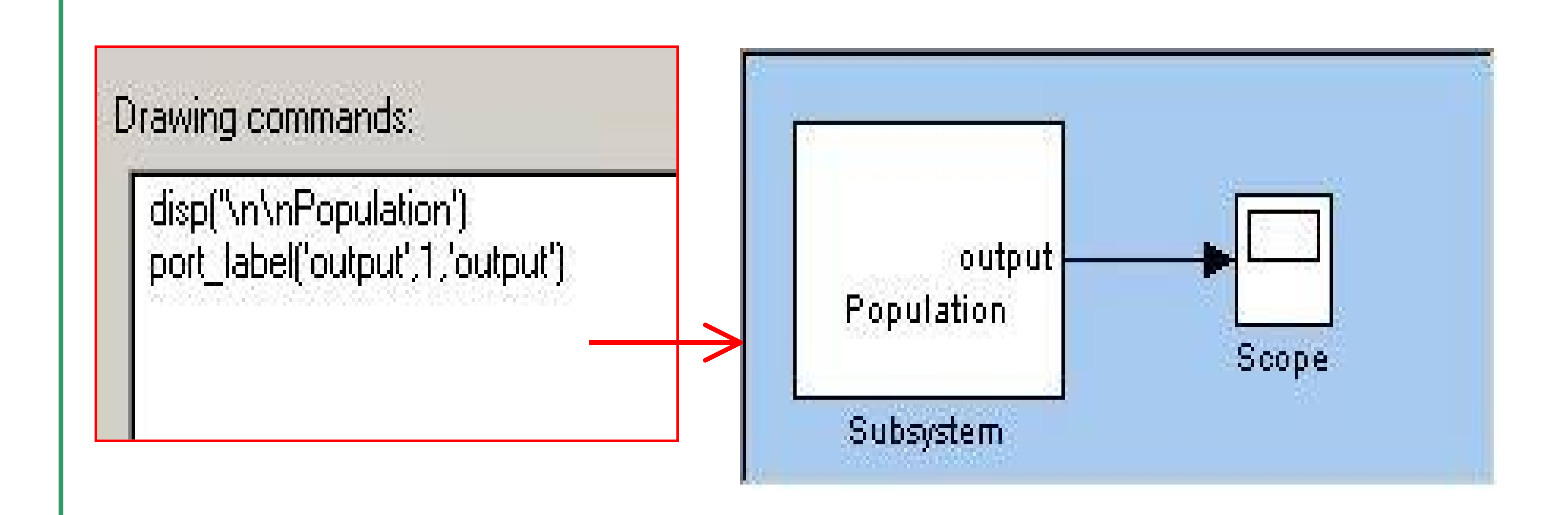


图7.19 图标绘制为文本

2) 图标为系统状态方程

使用dpoly命令在设置封装后子系统模块的图标为系统传递函数，或采用droots命令设置为系统零极点传递函数，其命令格式为

dpoly(num, den)

dpoly(num, den, 'character')

droots(z, p, k)

其中num、den分别为分子与分母多项式，'character'（如s或是z）为系统频率变量，z、p、k分别为零点、极点与系统增益。需要注意的是，num、den、z、p、k均为MATLAB工作空间中已经存在的变量，否则绘制命令的执行将出现错误。绘制封装后子系统的图标为系统传递函数如图7.20所示。num=[1]，den=[1 2 1]为MATLAB中的变量。

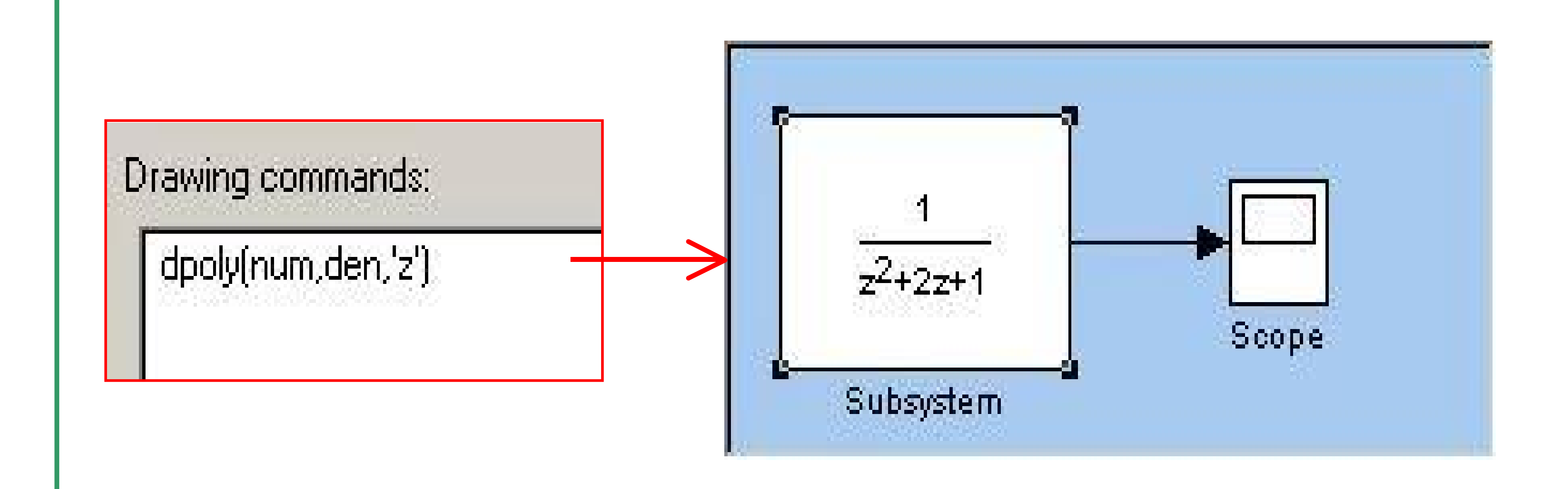


图7.20　图标绘制为系统传递函数

3) 图标为图像或图形

使用plot命令与image命令可以设置封装后子系统模块的图标为图形或图像。尽管一般的MATLAB命令不能在图标绘制命令栏中直接使用，但它们的返回值可以作为图标绘制命令的参数。绘制封装后子系统模块的图标为图形或图像，如图7.21所示。

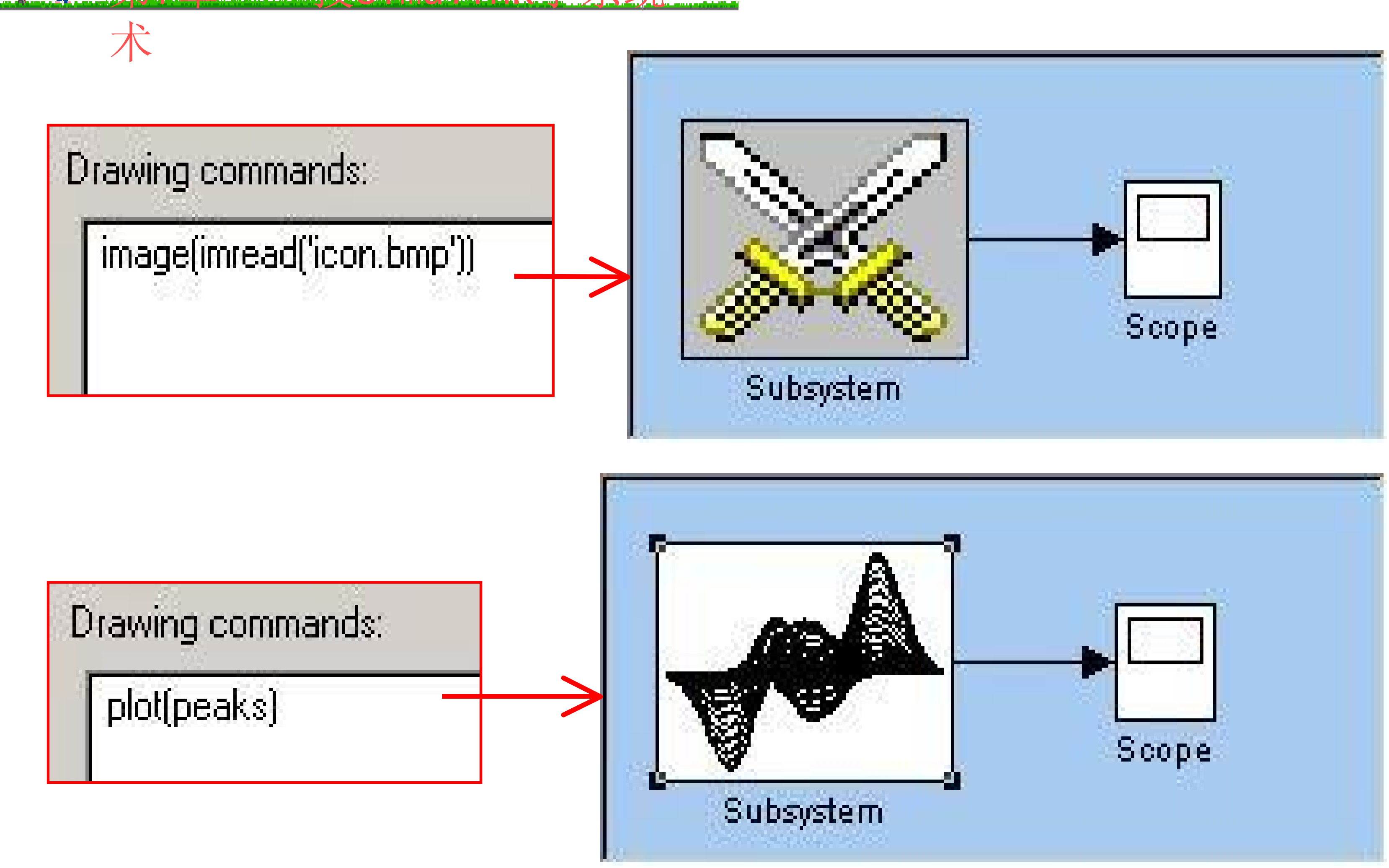


图7.21　图标绘制为图像与图形

7.3.3 封装编辑器之参数初始化对话框

子系统封装最主要的目的之一便是提供一个友好的参数设置界面。一般的用户无需了解系统内部实现，只需提供正确的模块参数，以使用特定模块的特定功能，从而完成系统设计与仿真分析的任务。如果只是绘制了模块的图标，则模块并没有被真正封装，因为在双击模块时仍显示模块内部的内容，并且始终直接使用来自MATLAB工作空间中的参数。

对通常的子系统与封装后的子系统作一个简单的比较：

(1) 通常的子系统可以视为MATLAB脚本文件，其特点是子系统没有输入参数，可以直接使用MATLAB工作空间中的变量。

(2) 封装后的子系统可以视为MATLAB的函数，其特点是封装后的子系统提供参数设置对话框输入参数；不能直接使用MATLAB工作空间中的变量；拥有独立的模块工作区（工作空间）；包含的变量对其它子系统及模块不可见；可以在同一模型中使用同样的子系统而其取值可各不相同。

在【例7.3】中已经介绍了如何将指定的子系统进行封装并进行图标绘制，这里仍以人口变化的非线性离散系统模型为例说明如何对封装后的子系统的参数输入对话框进行设置。在这个人口变化的简单模型中，系统的动力学方程由如下的差分方程来描述：

$$p(n)=rp(n-1)\left[1-\frac{p(n-1)}{K}\right]$$

其中表示某一年的人口数量，表示人口繁殖速率，表示新增资源所能满足的个体数目。因此在封装后子系统模块的参数设置界面中，应该提供相应的初始参数、繁殖速率以及。

Mask Editor: population_system/Subsystem

Icon | Initialization | Documentation

Mask type: Population Sample Mask

Prompt	Type	Variable
Initial Population:	edit	p
Propagate Rate:	edit	r
Individual Number:	edit	k
<<end of parameter list>>		

Add
Delete
Up
Down

Prompt: Initial Population:
Control type: Edit
Variable: p
Assignment: Evaluate
Popup strings:
Initialization commands:

OK Cancel Unmask Help Apply

参数设置控制
参数描述
参数对应变量
参数设置列表
参数设置描述
初始化命令

图7.22 子系统模块参数设置（参数初始化）

下面对参数初始化选项卡的内容作逐一介绍。

1. 参数设置控制

参数设置控制包括添加（Add）、删除（Delete）、上移（Up）与下移（Down），它们分别表示在即将生成的参数设置对话框中添加、删除、上移与下移模块需要的输入参数。

2. 参数描述（Prompt）

参数描述指的是对模块输入的参数作简单的说明，其取值最好能够说明参数的意义或者作用。

3. 参数对应变量（Variable）

参数对应变量表示键入的参数值将传递给封装后的子系统工作空间中相应的变量，在此使用的变量必须与子系统中所使用的变量具有相同的名称。

4. 参数设置描述

参数设置描述包括参数控制类型（Control type）、参数分配类型（Assignment）以及下拉选项框（Popup strings）。其中控制类型包括Edit（需要用户键入参数值，适合多数情况）、Checkbox（复选框，表示逻辑值）及Popup（弹出参数选项以供选择取值，弹出参数选项用Popup strings栏中由'|'隔开的字符串表示）。

5. 初始化命令栏（Initialization commands）

初始化命令为一般的MATLAB命令，在此可以定义封装后子系统工作空间中的各种变量，这些变量可以被封装子系统模块图标绘制命令、其它初始化命令或子系统中的模块使用。当出现下述情况时，Simulink开始执行初始化命令：

(1) 模型文件被载入。

(2) 框图被更新或模块被旋转。

(3) 绘制封装子系统模块图标时。

在参数设置对话框中输入正确的参数，设置人口的初始值为p＝100 000、人口繁殖速率为r＝1.05，而新增资源所能满足的个体数目K＝1 000 000。然后单击Apply或OK按钮并采用与第5章中相同的仿真参数。运行系统，其仿真结果如图7.24所示。

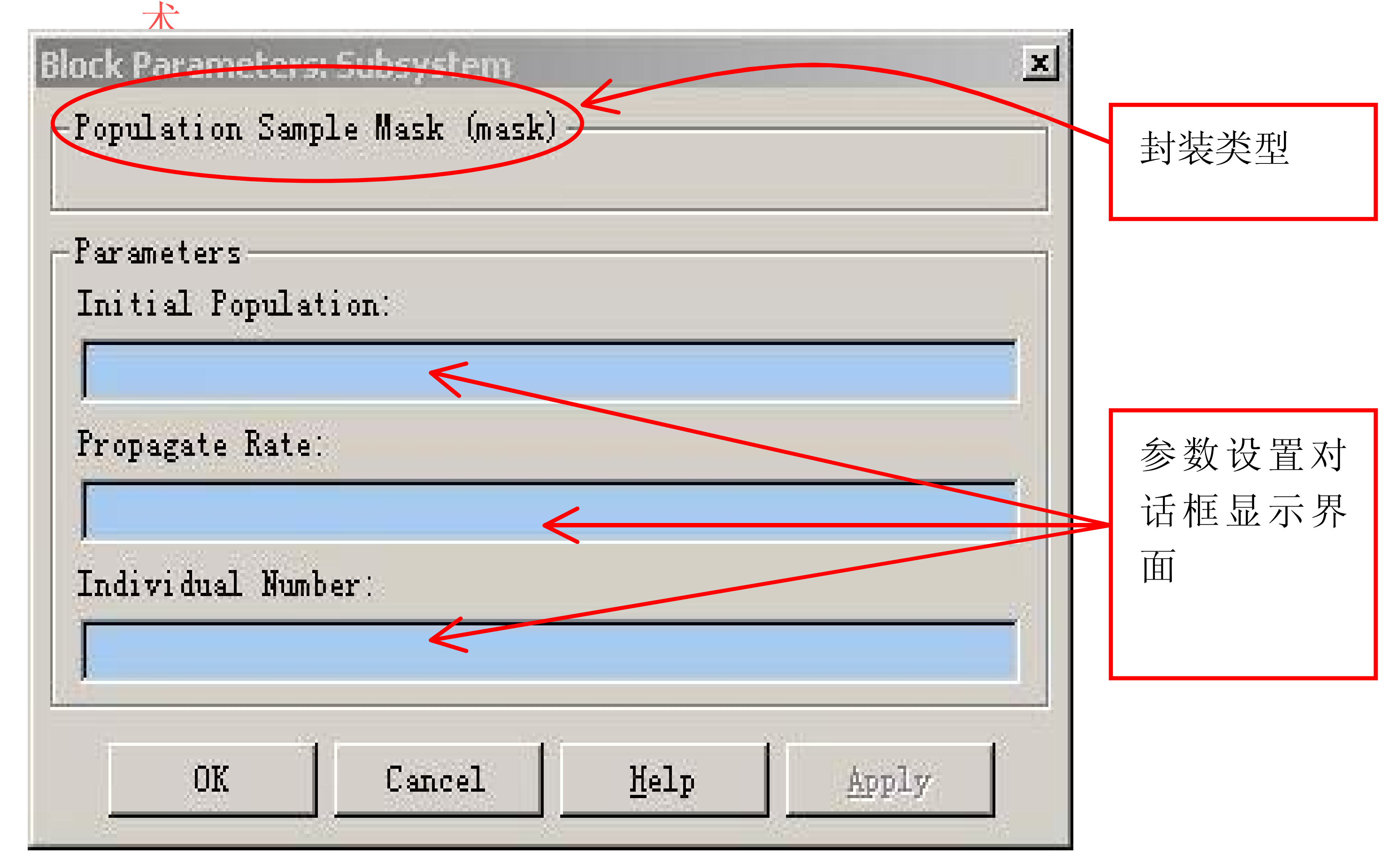
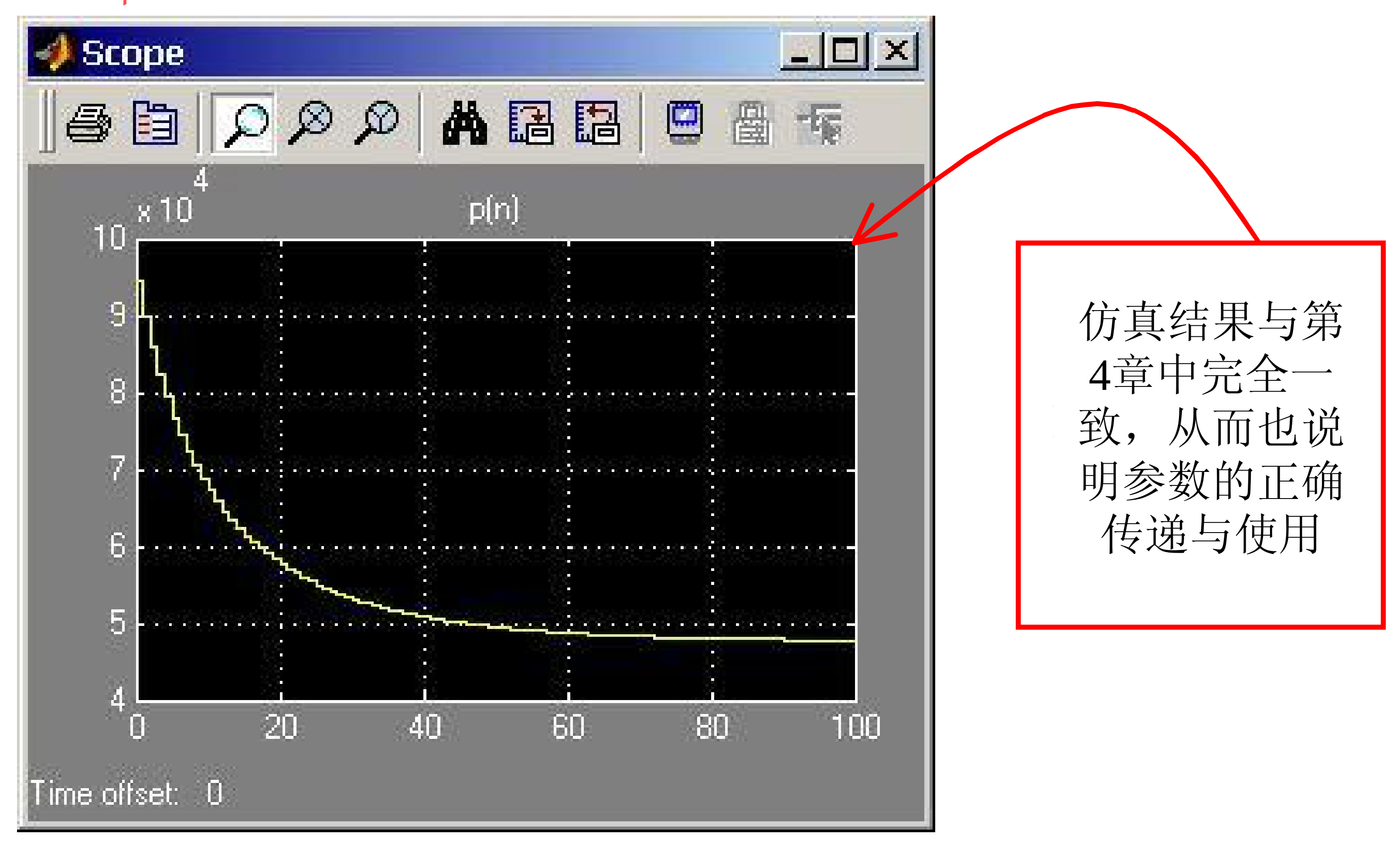


图7.23 封装后子系统的参数设置对话框



Scope
x 10^4
p(n)
10
9
8
7
6
5
4
0
20
40
60
80
100
Time offset: 0
仿真结果与第4章中完全一致，从而也说明参数的正确传递与使用

图7.24　系统仿真结果

7.3.4 封装编辑器之文档编辑对话框

Simulink模块库中的内置模块均提供了简单的描述与详细的帮助文档，这可以大大方便用户的使用与理解。对于用户自定义的模块（即封装后的子系统），Simulink提供的文档编辑功能同样可使用户建立自定义模块的所有帮助文档。图7.25所示为封装编辑器中文档编辑选项卡（Documentation），使用文档编辑可以建立用户自定义模块的简单描述文档与模块的详细帮助文档（包括模块的所有信息，可以使用HTML格式编写）。

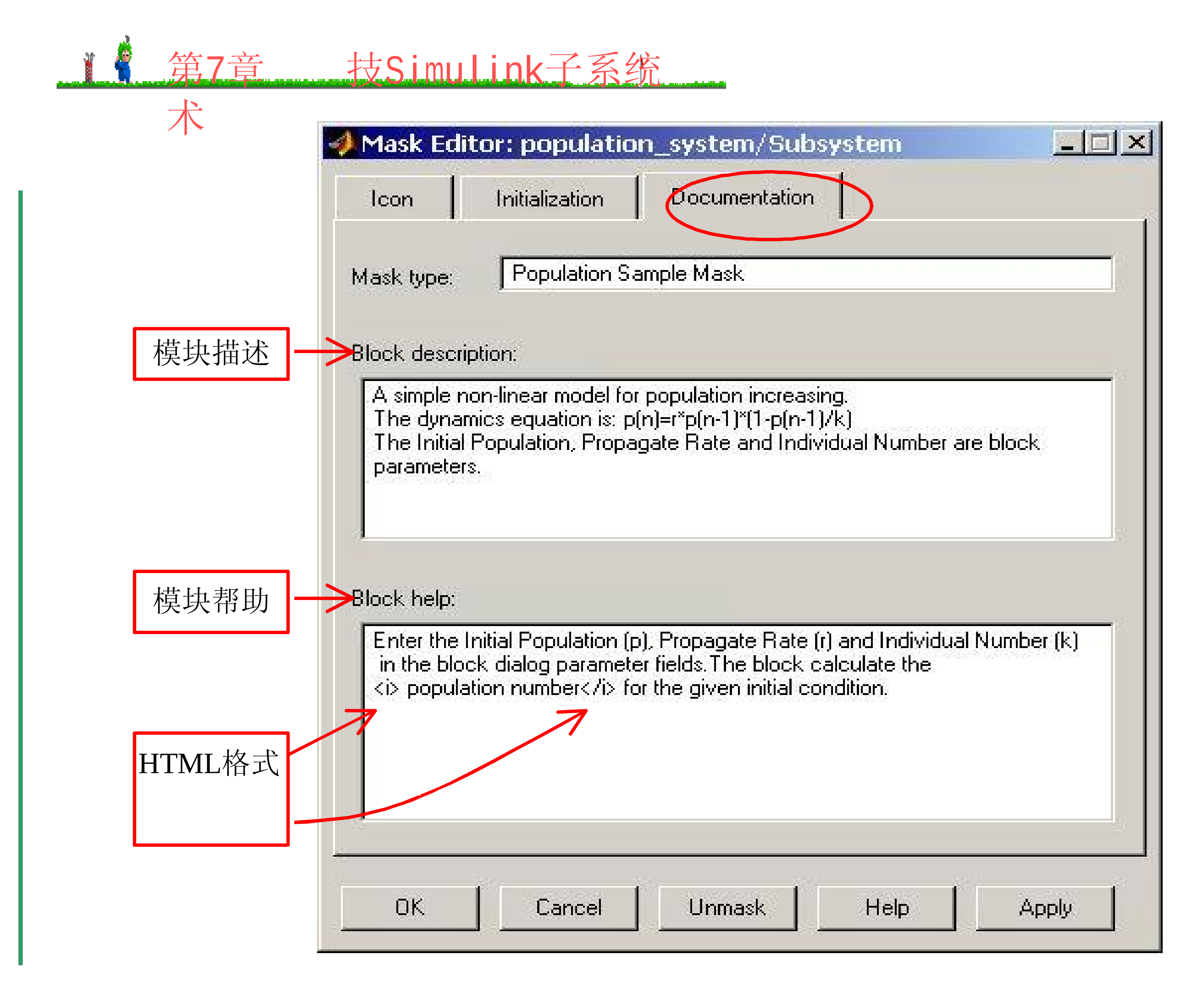


图7.25 封装编辑器的文档编辑

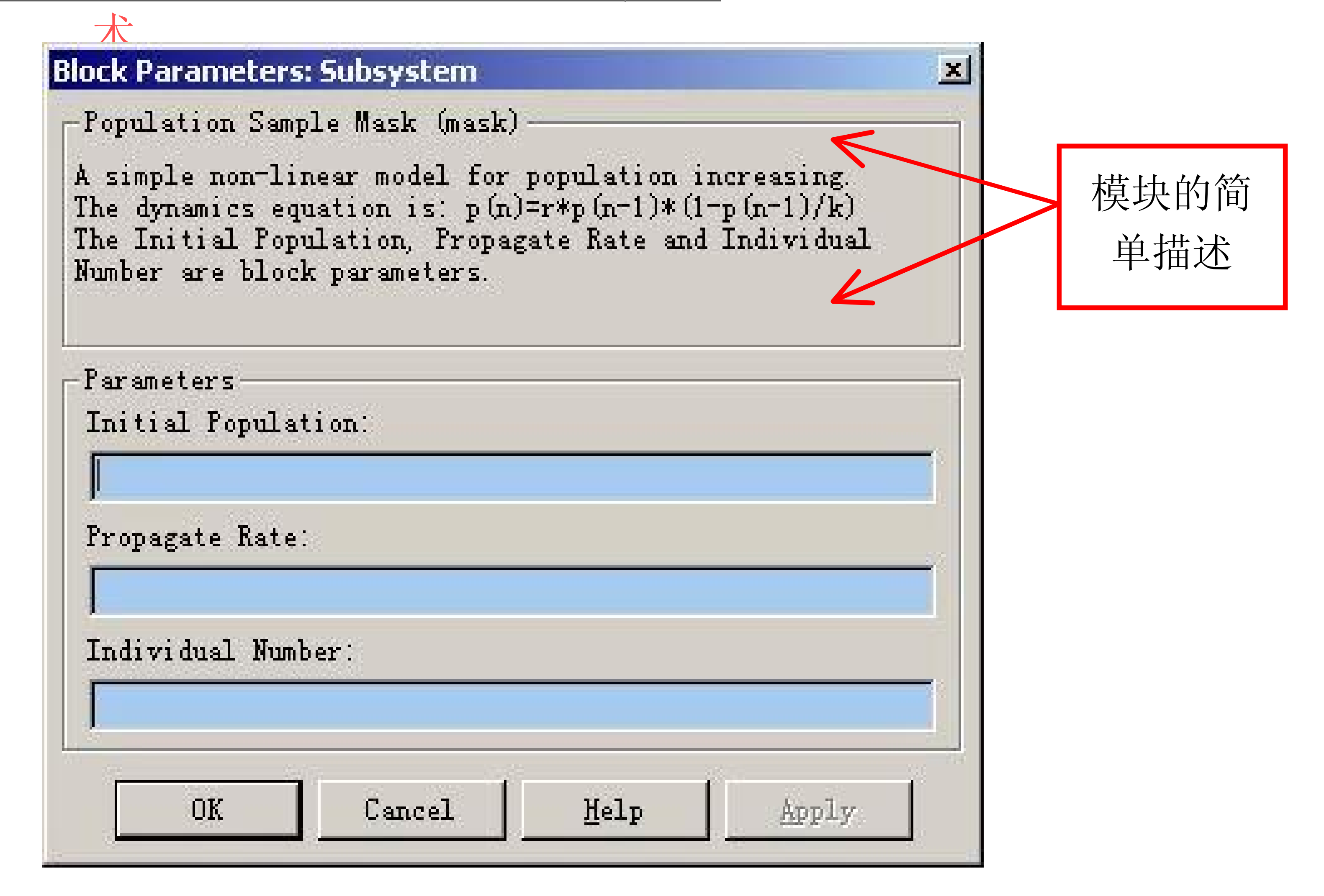


图7.26 带有模块简单描述的参数设置对话框

编写一个好的文档对于系统的设计与开发往往是至关重要的，它便于用户对系统的使用与维护。如果这时单击Help帮助按钮，用户可以MATLAB中的帮助系统中获得模块的更进一步的说明与其它的所有相关信息，如图7.27所示。

至此，本章已经比较全面地介绍了子系统封装的概念及其使用，下面将进一步介绍使用Simulink自定义系统模块库的技术。

Help

File Edit View Go Web Window Help

Help Navigator

roduct filter: All Selected Select...

Contents Index Search Favor

Begin Here
Release Notes for Relea
Installation
MATLAB
Simulink
Using Simulink
Getting Started
Quick Start

Find in page: Go

Simulink Masked Block: Add to Favorites

Using Simulink Block Reference

Population Sample Mask

Enter the Initial Population (p), Propagate Rate (r) and Individual Number (k) in the block dialog parameter fields.The block calculate the *population number* for the given initial condition.

模块帮助文档

HTML格式

图7.27 子系统模块的帮助文档



7.4　Simulink模块库技术

Simulink使用模块库技术来组织与管理具有某种属性的同一类模块，这给用户带来了很大的方便。同时Simulink也给用户提供了定制模块库以组织、管理用户自定义的模块。本节将详细介绍如何使用Simulink定制自定义的模块库。

7.4.1 模块库的概念及其应用

所谓的模块库一般是指具有某种属性的一类模块的集合。Simulink的库浏览器中包含了大量的用于不同领域的模块库，用户可以使用其中的任何模块来建立自己的动态系统模型并进行动态仿真与分析。

说明以下几个常用的术语：

(1) 模块库：具有某种属性的一类模块的集合。

(　　(2) 库模块：模块库中的一个模块。

(3) 引用块：模块库中的一个模块的副本（从模块库中拖动或复制到系统模型中的模块）。

(4) 关联：引用块与对应的模块库中的模块之间的联系，当模块库中的模块发生改变时Simulink会自动更新相应的引用块。

7.4.2 建立与使用模块库

建立Simulink模块库的方法如下：

(1) 在Simulink中使用File菜单下的New\Library建立一个新的模块库。

(2) 将用户自定义的模块或是其它模块库中的模块移动到新的模块库中。

(3) 保存新的模块库。

下面举例说明。首先建立一个新的模块库（其名称为MyOwnLib），然后将在7.3节中封装后的人口动态系统模块移动到新库之中并保存，最后使用此模块库中的模块建立如图7.28所示的动态系统。

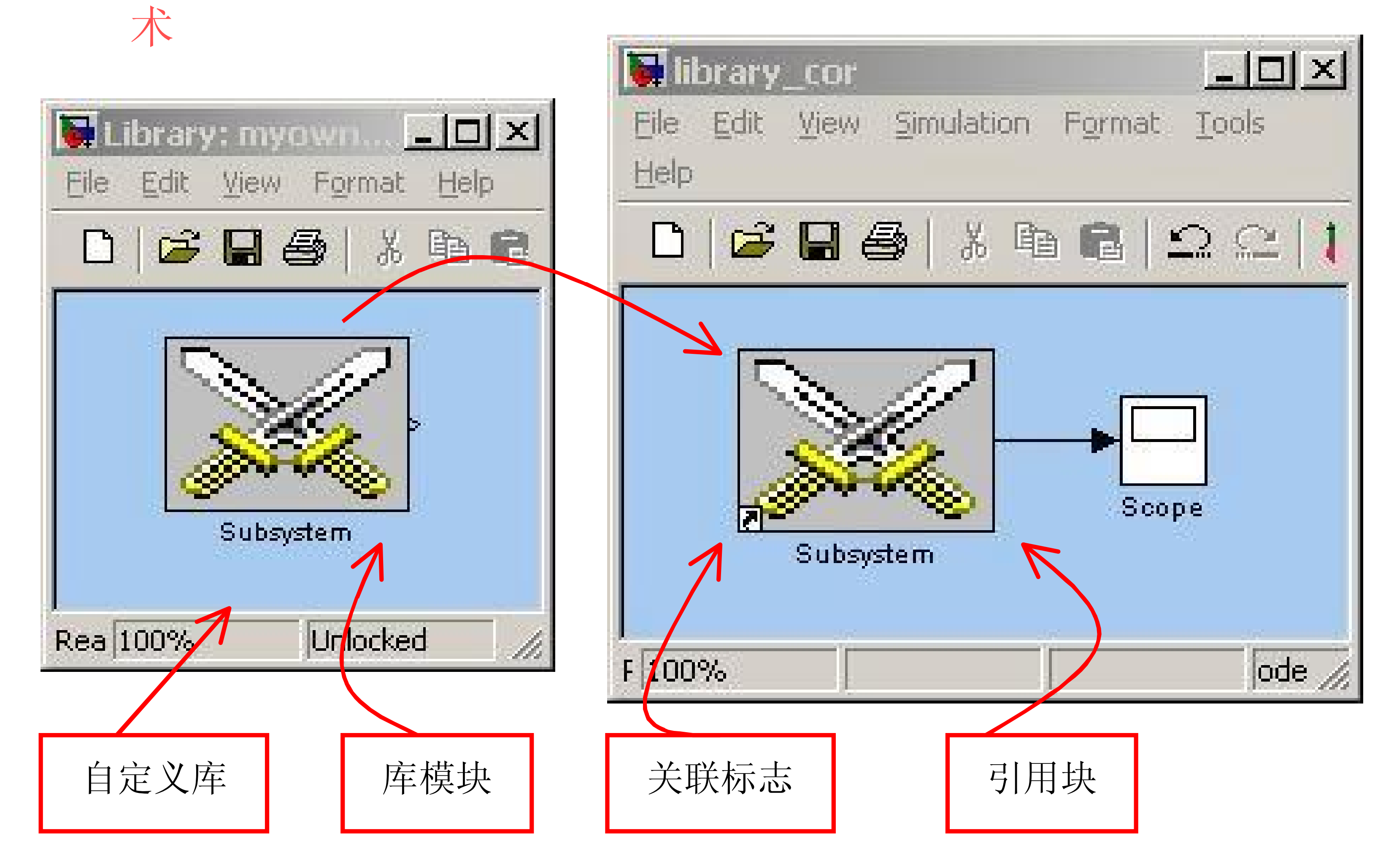


图7.28 使用自定义模块库建立动态系统

此时双击系统模型中的引用块，则会弹出与7.3节中相同的参数设置对话框，只要用户输入正确的参数并设置合适的仿真参数，其仿真结果就会与7.3节中的完全一致。

当一个新建立的模块库关闭之后，模块库便被锁住了。用户可以使用Edit菜单下的Look under mask来查看模块库中选定模块的内部实现。



图7.29 模块库修改确认对话框

如果需要对引用块的内容进行修改，可采用如下两种方法：

(1) 用鼠标右键单击块，选择Link Options下的Go to library block命令，对与引用块相应的模块库进行解锁，修改模块库中的模块。然后使用Edit菜单下的Update diagram以更新模型框图，但是这会影响到所有引用块。

(2) 用鼠标右键单击块，选择Link Options下的Disable link命令（这时关联标志箭头将变为灰色）。然后用鼠标右键单击模块，选择Look under mask，打开模块以修改模型中模块的内容。这只影响当前的块。

7.4.3 库模块与引用块的关联

Simulink通过关联来表示模块库中的模块与相应的引用块之间的关系。为了使用户对关联有一个更深入的了解，这里以一个简单的例子来说明关联是如何工作的。

【例7.4】 关联的使用。其步骤如下：

(1) 建立一个新的系统模型，从自定义的模块库MyOwnLib中拖动自定义的系统模块（人口变化动态系统）到此模型框图中。图7.30所示为在新建立的系统模型中同时使用两个引用块。

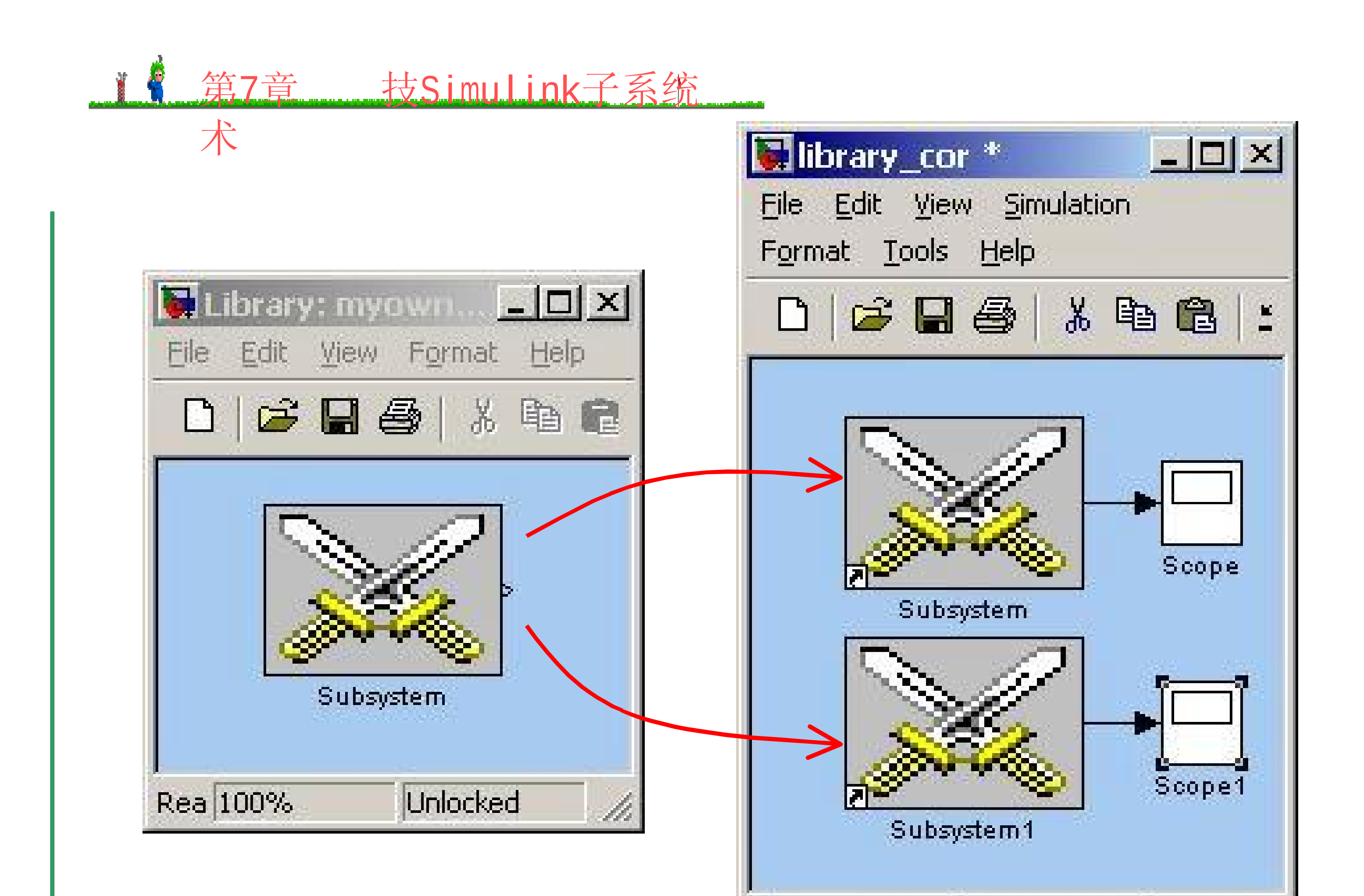


图7.30 在系统模型框图中使用两个引用块

(2) 断开第二个引用块与库模块的关联：用鼠标右键单击引用块，选择Link Options下的Disable link命令即可。

(3) 修改模块库中的模块，在子系统中加入注释文本“This is the modified library model”。用鼠标右键单击并选择Look under mask命令以修改，如图7.31所示。

(4) 使用Edit下的Update diagram刷新系统模型框图，然后使用Look under mask查看引用块的变化，则第一个模块的内容随模块库的改变而改变，而第二个模块并没发生任何变化，如图7.32所示。

(5) 修改禁止关联的第二个引用块，在其中加入注释“Disable Link”，然后保存系统模型并试图恢复关联。用鼠标右键单击第二个引用块，选择Link options下的Restore link命令，此时Simulink将弹出如图7.33所示的恢复连接对话框。

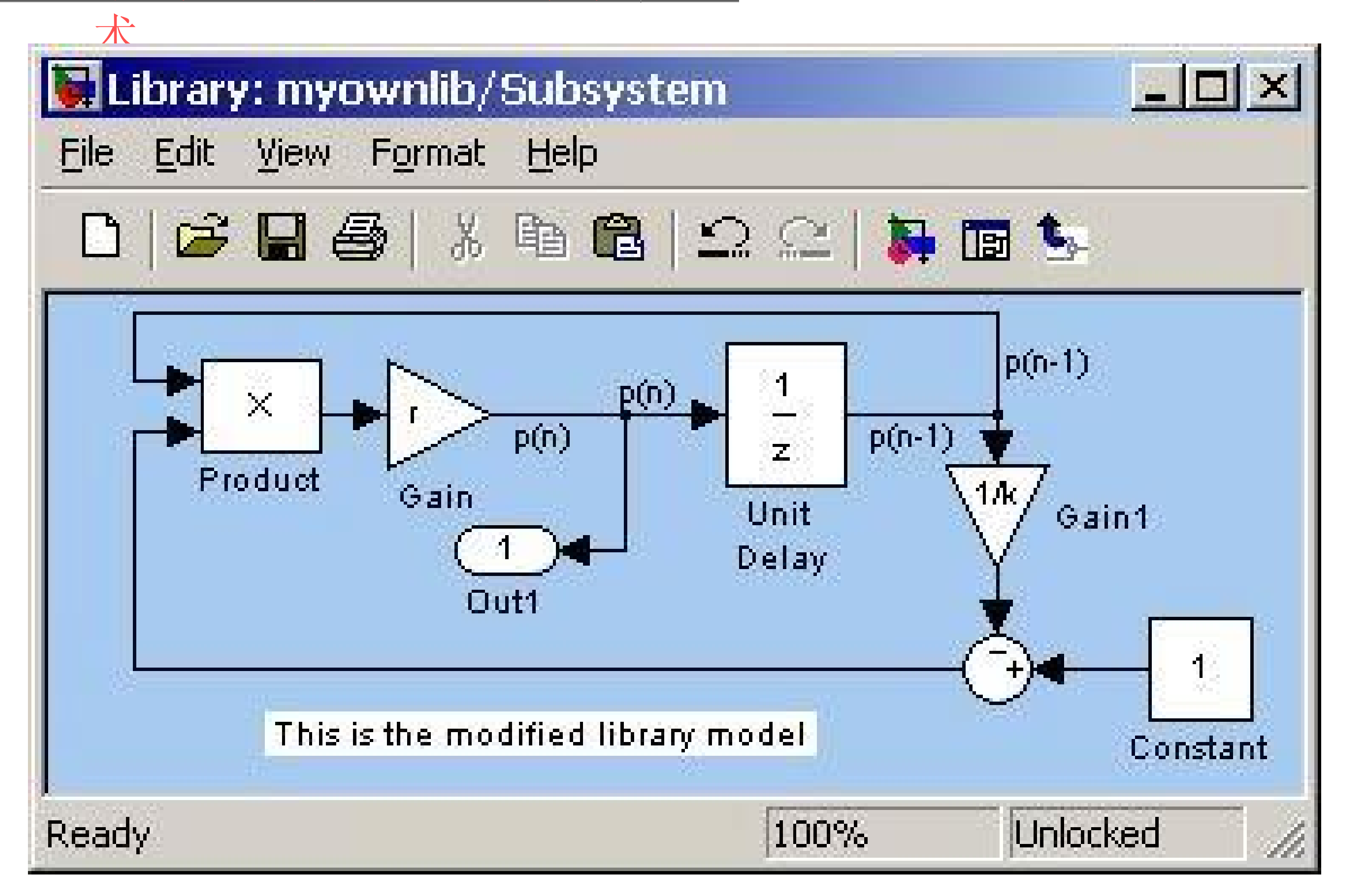


图7.31 对模块库中的模块进行修改

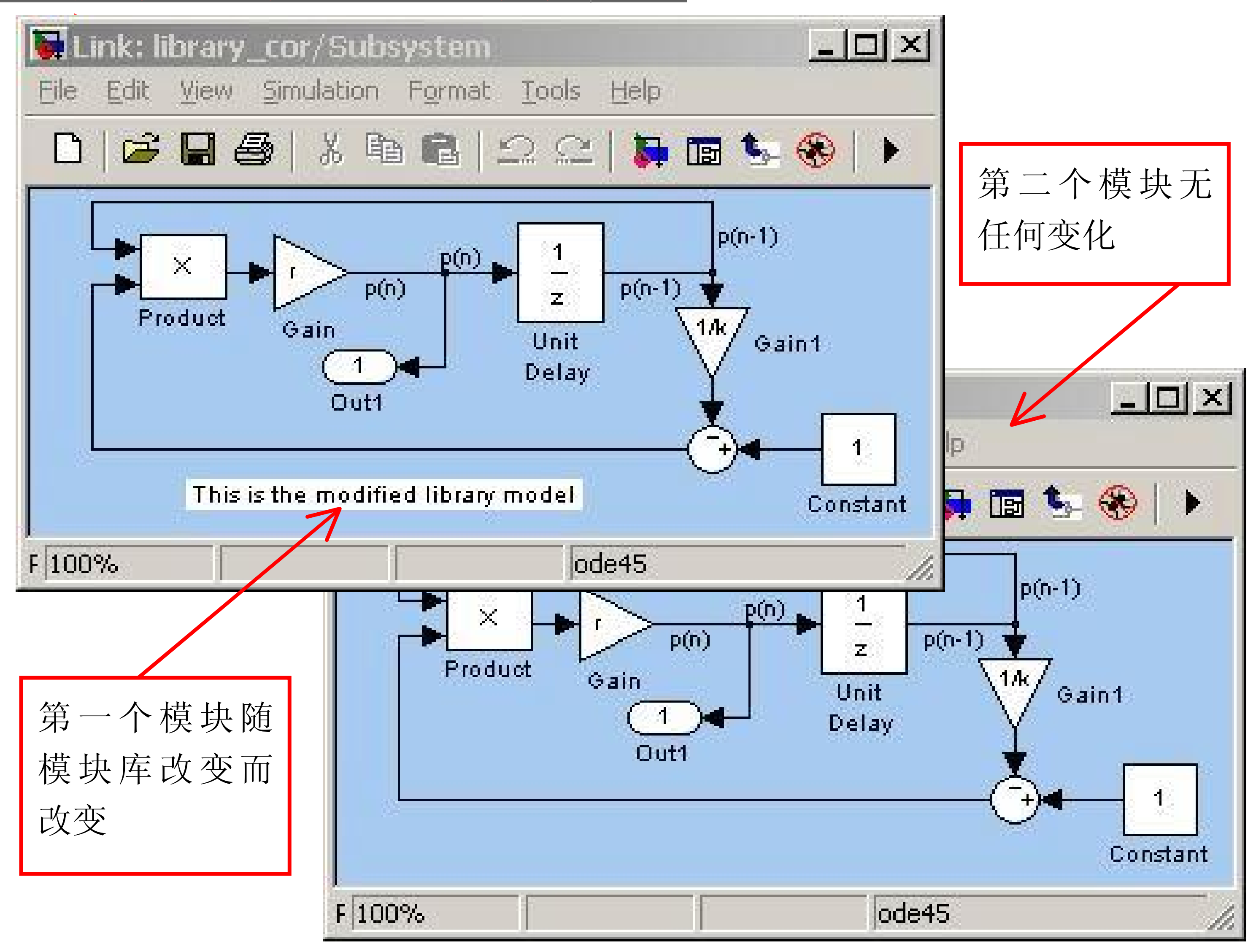


图7.32 修改模块库中模块对不同引用块的影响

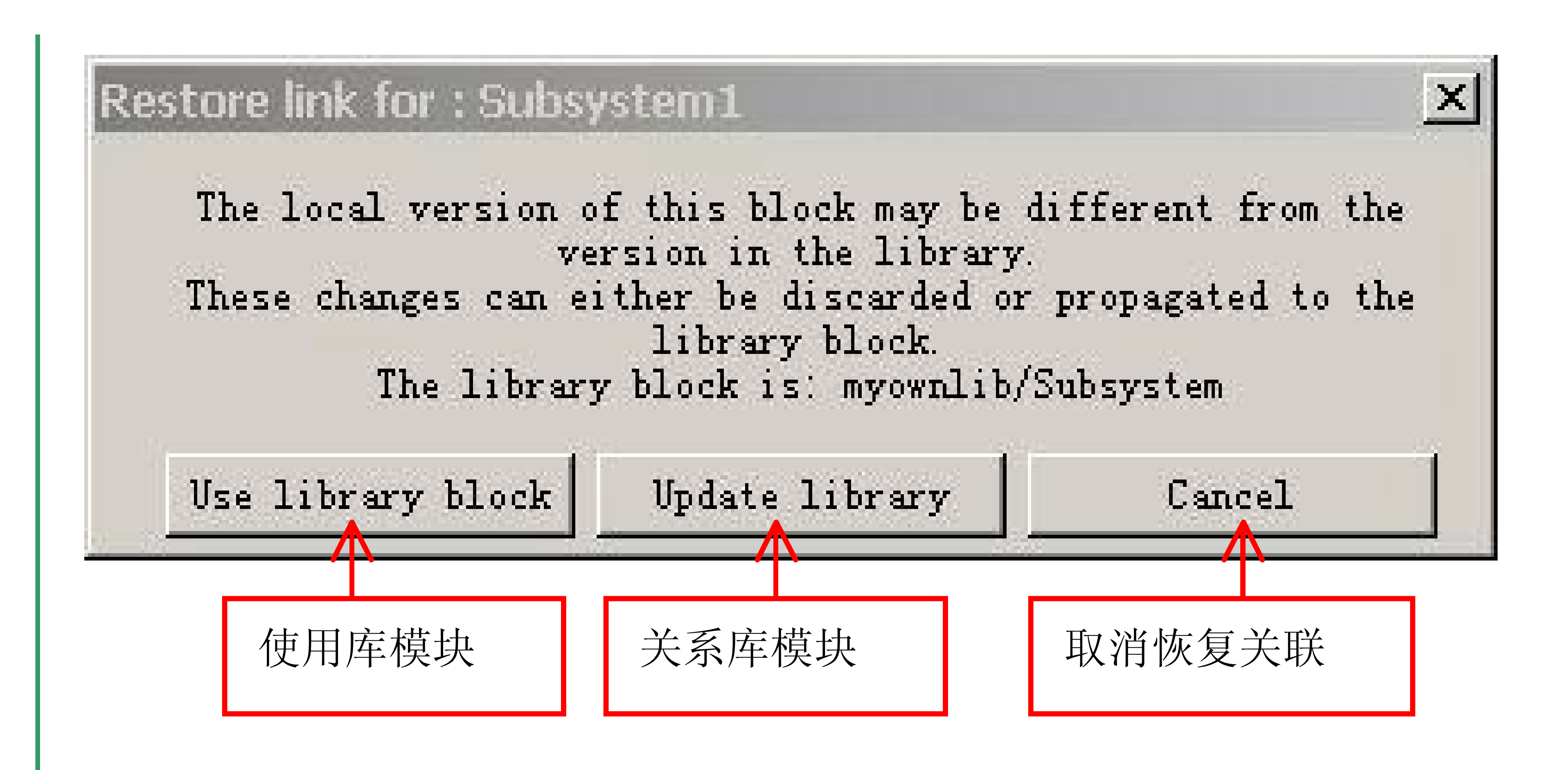


图7.33 恢复连接对话框

7.4.4 可配置子系统

7.2节中介绍Simulink的子系统时指出，可配置子系统只能够在用户自定义的模块库使用。在某些情况下，用户建立的系统模型中可能有若干个不同的子系统，它们具有同样的功能。如果用户需要在它们之间频繁的切换，便可为这些子系统建立可配置子系统。建立与使用可配置子系统的具体方法如下：

(1) 建立包含这些子系统的自定义模块库，然后从Subsystems模块库中拖动Configurable Subsystem 模块到这个自定义模块库中，如图7.34所示。

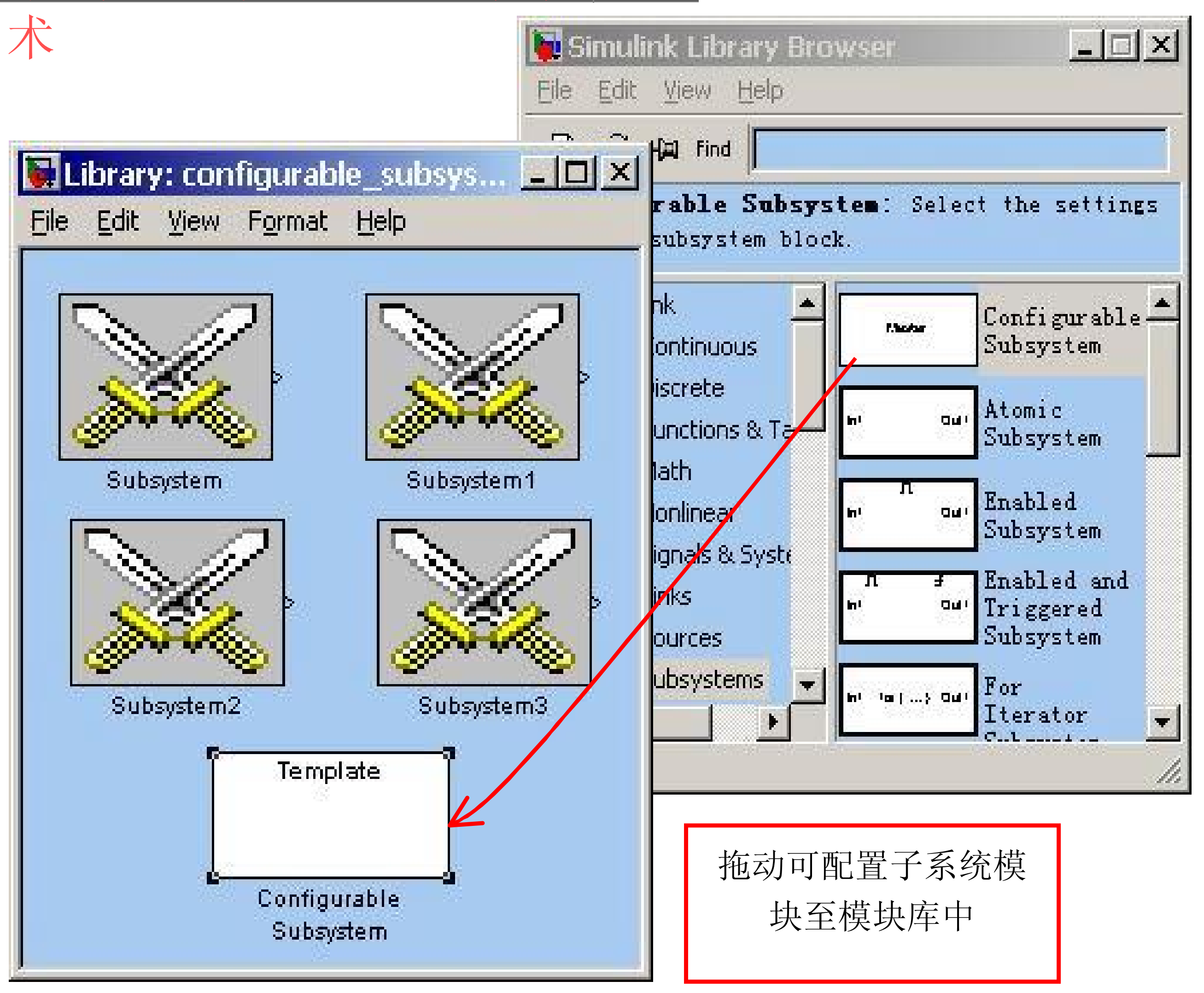


图7.34 建立可配置子系统

(2) 保存自定义模块库，然后双击可配置子系统块。选择用户需要相互切换的子系统，如图7.35所示。

(3) 复制模块库中的可配置子系统到相应的系统模型中，如图7.36所示。

(4) 使用Edit下的Block Choice项完成模块的配置。

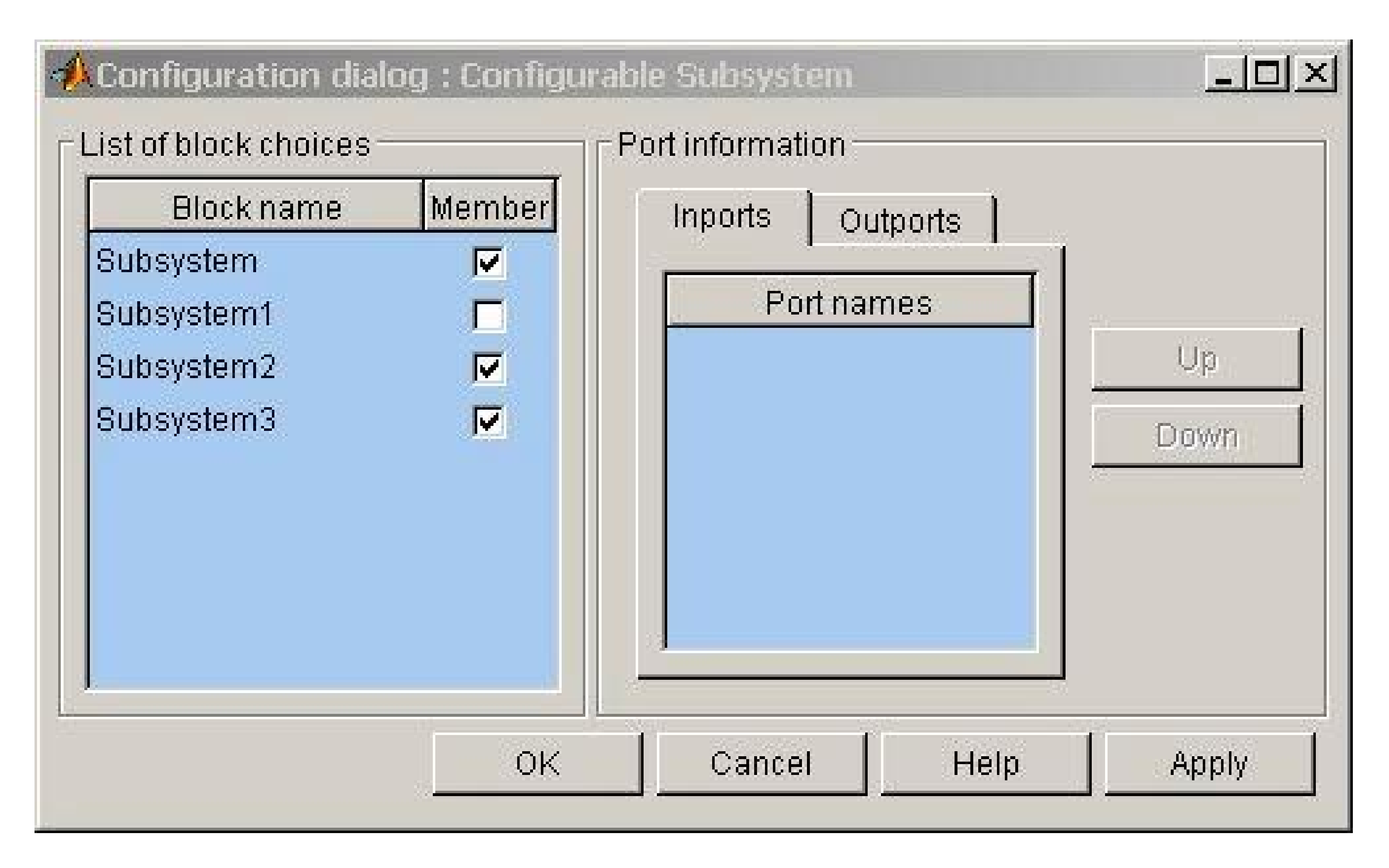


图7.35 选择需要使用的子系统

图7.36 在系统模型中使用可配置子系统



第8章 Simulink命令行仿真技术

8.1 使用命令行方式建立系统模型

除了使用Simulink的图形建模方式建立动态系统模型之外，用户也可以使用命令行方式进行系统建模，然后再进行动态系统的仿真与分析。在进一步介绍使用命令行进行动态系统的仿真技术之前，首先简单介绍一下使用命令行的方式建立系统模型的相关知识。Simulink中建立系统模型的命令如表8.1所示。

表8.1　系统模型建立命令

命　令	功　能
new_system	建立一个新的Simulink系统模型
open_system	打开一个已存在的Simulink系统模型
close_system, bdclose	关闭一个Simulink系统模型
save_system	保存一个Simulink系统模型
find_system	查找Simulink系统模型、模块、连线及注释
add_block	在系统模型中加入指定模块
delete_block	从系统模型中删除指定模块
replace_block	替代系统模型中的指定模块
add_line	在系统模型中加入指定连线

delete_line	从系统模型中删除指定连线
get_param	获取系统模型中的参数
set_param	设置系统模型中的参数
gcb	获得当前模块的路径名
gcs	获得当前系统模型的路径名
gcbh	获得当前模块的操作句柄
bdroot	获得最上层系统模型的名称
simulink	打开Simulink的模块库浏览器

使用上面的命令便可以生成和编辑动态系统的Simulink模型，由于使用命令行方式建立的Simulink系统模型与使用图形建模方式建立的系统模型没有什么大的分别，因此这里仅简单介绍各个命令的使用，而不再给出使用这些命令所建立的系统模型框图。

1. new_system

1) 使用语法

```
new_system('sys')
```

2) 功能描述

使用给定的名称建立一个新的Simulink系统模型。如果'sys'为一个路径，则新建立的系统为在此路径中指定的系统模型下的一个子系统。注意，new_system命令并不打开系统模型窗口。

3) 举例

new_system('mysys')% 建立名为'mysys'的系统模型

new_system('vdp/mysys')% 建立系统模型vdp下的子系统'mysys'

2. open_system

1) 使用语法

```
open_system('sys')
open_system('blk')
open_system('blk', 'force')
```

2) 功能描述

打开一个已存在的Simulink系统模型。

open_system('sys')：打开名为'sys'的系统模型窗口或子系统模型窗口。注意，这里'sys'使用的是MATLAB中标准路径名（绝对路径名或相对于已经打开的系统模型的相对路径名）。

open_system('blk')：打开与指定模块'blk'相关的对话框。open_system('blk', 'force')：打开封装后的子系统，这里'blk'为封装子系统模块的路径名。这个命令与图形建模方式中的Look under mask菜单功能一致。

3) 举例

open_system('controller')　　% 打开名为controller的系统模型

open_system('controller/Gain')　　% 打开controller模型下的增益模块Gain的对话框

3. save_system

1) 使用语法

```
save_system
save_system('sys')
save_system('sys', 'newname')
```

2) 功能描述

保存一个Simulink系统模型。

save_system：使用当前名称保存当前顶层的系统模型。

save_system('sys')：保存已经打开的系统模型，与save_system功能类似。

save_system('sys', 'newname')：使用新的名称newname保存当前已经打开的系统模型。

3) 举例

save_system　　　　　　% 保存当前的系统模型

save_system('vdp')　　% 保存系统模型vdp

save_system('vdp', 'myvdp')% 保存系统模型vdp，模型文件名为myvdp

4. close_system, bdclose

1) 使用语法

close_system

close_system('sys')

close_system('sys', saveflag)

close_system('sys', 'newname')

close_system('blk')

bdclose；bdclose('sys')；bdclose('all')

2) 功能描述

关闭一个Simulink系统模型。

close_system：关闭当前系统或子系统模型窗口。如果顶层系统模型被改变，系统会提示是否保存系统模型。

close_system('sys')：关闭指定的系统或子系统模型窗口。

close_system('sys', saveflag)：关闭指定的顶层系统模型窗口并且从内存中清除。saveflag为0表示不保存系统模型，为1表示使用当前名称保存系统模型。

3) 举例

close_system   % 关闭当前系统

close_system('engine', 1)   % 保存当前系统模型engine（使用当前系统名称），然后再关闭系统

close_system('engine/Combustion/Unit Delay')

% 关闭系统模型engine下的Combustion 子系统中Unit Delay模块的对话框

5. find_system

1) 使用语法

find_system(sys, 'c1', cv1, 'c2', cv2,...'p1', v1, 'p2', v2,...)

2) 功能描述

查找由sys指定的系统模型、模块、连线及注释等等，并返回相应的路径名与操作句柄。由于使用此命令涉及较多的参数设置，因此这里不再赘述，用户可以查看Simulink的联机帮助系统中Simulink目录下的Using Simulink\Model Construction Commands\Introduction中的find_system命令的帮助即可。

6. add_block

1) 使用语法

add_block('src', 'dest')

add_block('src', 'dest', 'parameter1', value1, ...)

2) 功能描述

在系统模型中加入指定模块。

add_block('src', 'dest')：拷贝模块'src'为'dest'（使用路径名表示），从而可以从Simulink的模块库中复制模块至指定系统模型中，且模块'dest'参数与'src'完全一致。

add_block('src', 'dest_obj', 'parameter1', value1, ...)：功能与上述命令类似，但是需要设置给定模块的参数'parameter1'，value1为参数值。

3) 举例

```
add_block('simulink3/Sinks/Scope', 'engine/timing/Scope1')
```

% 从Simulink的模块库Sinks中复制Scope模块至系统模型engine中子系统timing中，其名称

% 为Scope1

7. delete_block

1) 使用语法

```
delete_block('blk')
```

2) 功能描述

从系统模型中删除指定模块。

delete_block('blk')：从系统模型中删除指定的系统模块'blk'。

3) 举例

```
delete_block('vdp/Out1')          %从vdp模型中删除模块Out1
```

8. replace_block

1) 使用语法

replace_block('sys', 'blk1', 'blk2', 'noprompt')

replace_block('sys', 'Parameter', 'value', 'blk', ...)

2) 功能描述

替代系统模型中的指定模块。

replace_block('sys', 'blk1', 'blk2')：在系统模型'sys'使用模块'blk2'取代所有的模块'blk1'。如果'blk2'为Simulink的内置模块，则只需要给出模块的名称即可，如果为其它的模块，必须给出所有的参数。如果省略'noprompt'，Simulink会显示取代模块对话框。

replace_block('sys', 'Parameter', 'value', ..., 'blk')：取代模型'sys'中具有特定取值的所有模块'blk'。'Parameter'为模块参数，'value'为模块参数取值。

3) 举例

```
replace_block('vdp','Gain','Integrator','noprompt')
```

% 使用积分模块Integrator取代系统模型vdp中所有的增益模块Gain，并且不显示取代对话框

9. add_line、delete_line

1) 使用语法

```
h = add_line('sys','oport','iport')
h = add_line('sys','oport','iport', 'autorouting','on')
delete_line('sys', 'oport', 'iport')
```

2) 功能描述

在系统模型中加入或删除指定连线。

add_line('sys', 'oport', 'iport')：在系统模型'sys'中给定模块的输出端口与指定模块的输入端口之间加入直线。'oport'与'iport'分别为输出端口与输入端口（包括模块的名称、模块端口编号）。

add_line('sys','oport','iport', 'autorouting','on')：与add_line('sys','oport','iport')命令类似，只是加入的连线方式可以由'autorouting'的状态控制：'on'表示连线环绕模块，而'off'表示连线为直线（缺省状态）。

delete_line('sys', 'oport', 'iport')：删除由给定模块的输出端口'oport'至指定模块的输入端口'iport'之间的连线。

3) 举例

```
add_line('mymodel','Sine Wave/1','Mux/1')
% 在系统模型'mymodel'中加入由正弦模块Sine Wave的输
  出至信号组合模块Mux第一个输入间
% 的连线
delete_line('mymodel', 'Sine Wave/1','Mux/1')
% 删除系统模型'mymodel'中由正弦模块Sine Wave的输出
  至信号组合模块Mux第一个输入间的
% 连线
```

10. set_param和get_param

1) 使用语法

```
set_param('obj', 'parameter1', value1, 'parameter2', value2, ...)
get_param('obj', 'parameter')
get_param( { objects }, 'parameter')
get_param(handle, 'parameter')
get_param(0, 'parameter')
get_param('obj', 'ObjectParameters')
get_param('obj', 'DialogParameters')
```

2) 功能描述

设置与获得系统模型以及模块参数。

set_param('obj', 'parameter1', value1, 'parameter2', value2, ...)：其中'obj'表示系统模型或其中的系统模块的路径，或者取值为0；给指定的参数设置合适的值，取值为0表示给指定的参数设置为缺省值。在仿真过程中，使用此命令可以在MATLAB的工作空间中改变这些参数的取值，从而可以更新系统在不同的参数下运行仿真。

get_param('obj', 'parameter')：返回指定参数取值。其中'obj'为系统模型或系统模型中的系统模块。

get_param({ objects }, 'parameter')：返回多个模块指定参数的取值，其中{ objects }表示模块的细胞矩阵（Cell）。

get_param(handle, 'parameter')：返回句柄值为handle的对象的指定参数的取值。

get_param(0, 'parameter')：返回Simulink当前的仿真参数或默认模型、模块的指定参数的取值。

get_param('obj', 'ObjectParameters')：返回描述某一对象参数取值的结构体。其中返回到结构体中具有相应的参数名称的每一个参数域分别包括参数名称（如Gain）、数据类型，如string以及参数属性如read-only等。

get_param('obj', 'DialogParameters')：返回指定模块对话框中所包含的参数名称的细胞矩阵。

由于参数选项很多，这里不再赘述,读者可以参考Simulink帮助中的Model and BlockParameters中的详细介绍。

3) 举例

set_param('vdp', 'Solver', 'ode15s', 'StopTime', '3000') % 设置系统模型'vdp'的求解器为'ode15s'，仿真结

% 束时间为3000 s

set_param('vdp/Mu', 'Gain', '1000') % 设置系统模型vdp中模块Mu中的增益模块Gain的取值为1000

set_param('vdp/Fcn', 'Position', [50 100 110 120]) % 设置系统模型vdp中的Fcn模块的位置为

set_param('mymodel/Zero-Pole','Zeros','[2 4]','Poles','[1 2 3]')

% 设置系统模型mymodel中零极点模块Zero-Pole的零点Zeros取值为[2 4]，而极点Poles取值为

% [1 2 3]

get_param('Mysys/Subsys/Inertia','Gain')

% 获得系统模型Mysys中子系统Subsys中的Inertia模块的增益值。其结果如下（假设此系统模型

% 以存在）

ans =

1.0000e-006

p = get_param('untitled/Sine Wave', 'DialogParameters')

% 获得系统模型untitled下正弦信号模块Sine Wave的参数设置对话框中所包含的参数名称，

% 结果如下

p =

'Amplitude'

'Frequency'

'Phase'

'SampleTime'

11.　gcb、gcs以及gcbh

1) 使用语法

```
gcb
gcb('sys')
gcs
gcbh
bdroot
bdroot('obj')
```

2) 功能描述

gcb：返回当前系统模型中当前模块的路径名。

gcb('sys')：返回指定系统模型中当前模块的路径名。

gcs：获得当前系统模型的路径名。

gcbh：返回当前系统模型中当前模块的操作句柄。

bdroot：返回顶层系统模型的名称。

bdroot('obj')：返回包含指定对象名称的顶层系统的名称，其中'obj'为系统或模块的路径名。

12. simulink

1) 使用语法

simulink

2) 功能描述

打开Simulink的模块库浏览器。

使用命令行方式建立的动态系统模型与使用Simulink图形建模方式所建立的系统模型完全一致，然而，仅仅使用命令行方式建立系统模型是一件麻烦且低效的方法。

8.2 回顾与复习：Simulink与MATLAB的接口

先建立一个非常简单的动态系统，其功能如下：

(1) 系统的输入为一单位幅值、单位频率的正弦信号。

(2) 系统的输出信号为输入信号的积分。

其要求是：

(1) 系统的输入信号由MATLAB工作空间中的变量提供，时间范围为0至10 s。

(2) 使用MATLAB绘制原始输入信号与系统运算结果的曲线。

从动态系统的功能与其要求可知，建立此动态系统的模型需要以下一些模块：

(1) Sources模块库中的In1模块：表示系统的输入。

(2) Sinks模块库中的Out1模块：表示系统的输出。

(3) Continuous模块库中的Integrator模块：积分运算器。

其中In1（Inport）模块与Out1（Outport）模块都是虚模块，它们仅仅表示将信号传入或传出子系统。

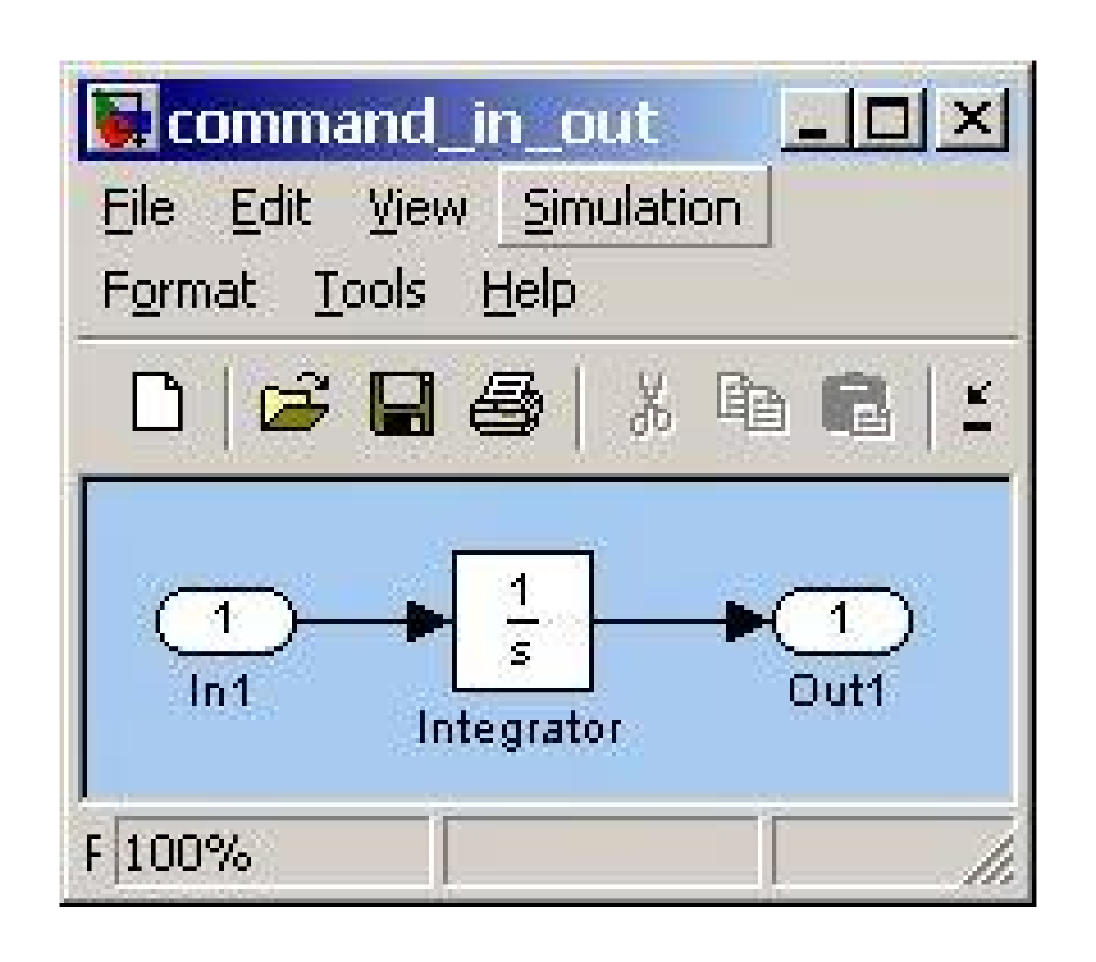


图8.1 简单动态系统模型框图

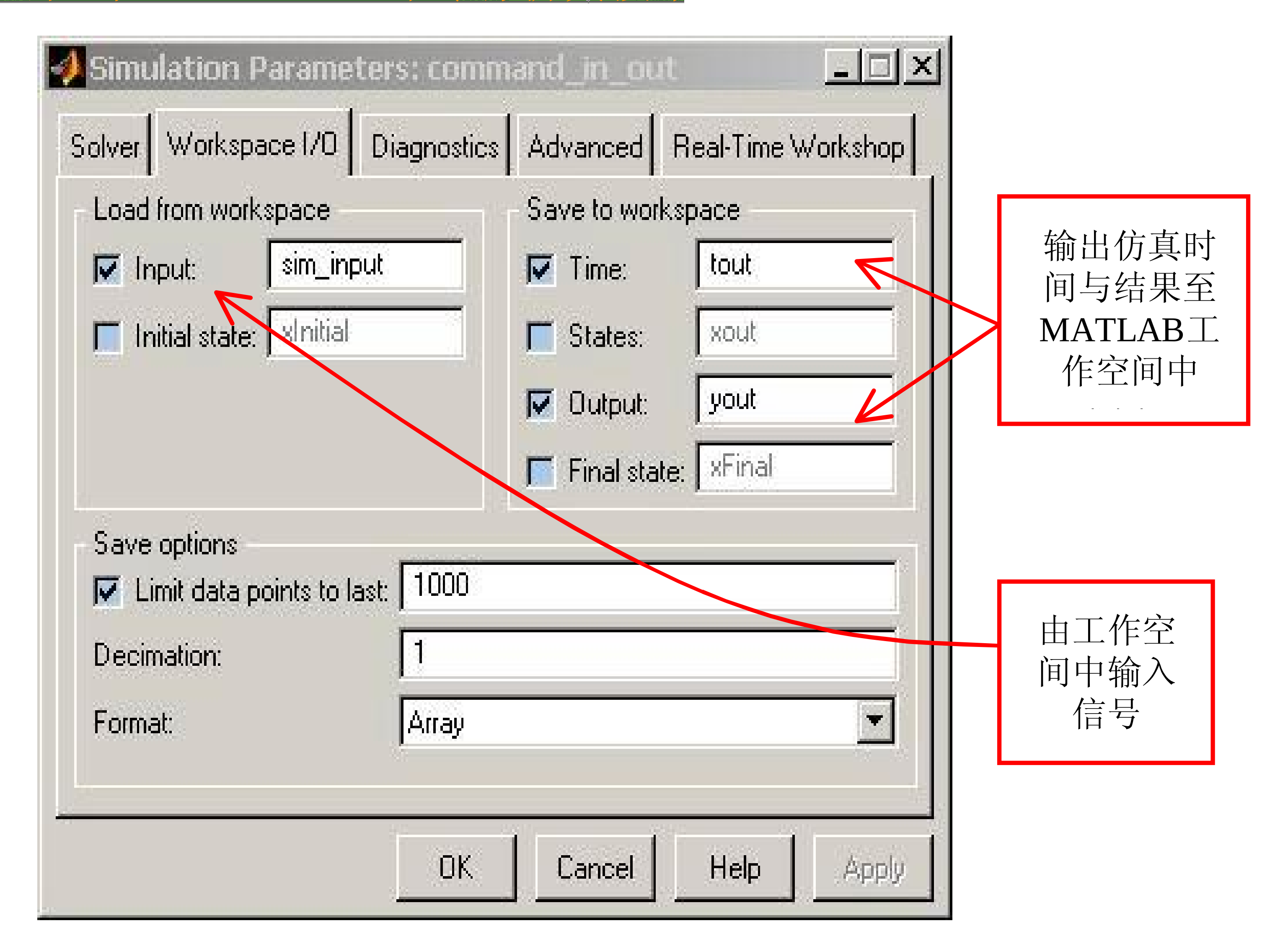


图8.2 Workspace I/O 设置

然后在MATLAB工作空间中定义输入变量sim_input如下：

```
>> t=0:0.1:10;                     % 表示输入信号的时间范围
>> u=sin(t);                       % 产生输入正弦信号
>> sim_input=[t',u'];              % 传递至Simulink系统模型的变量
```

接下来，采用默认的系统仿真参数并运行系统仿真。最后使用MATLAB命令绘制出原始输入信号与系统运算结果，如下所示：

>> plot(t,u,tout,yout,‘--’) %绘制系统输入信号与仿真结果，如图8.3所示.

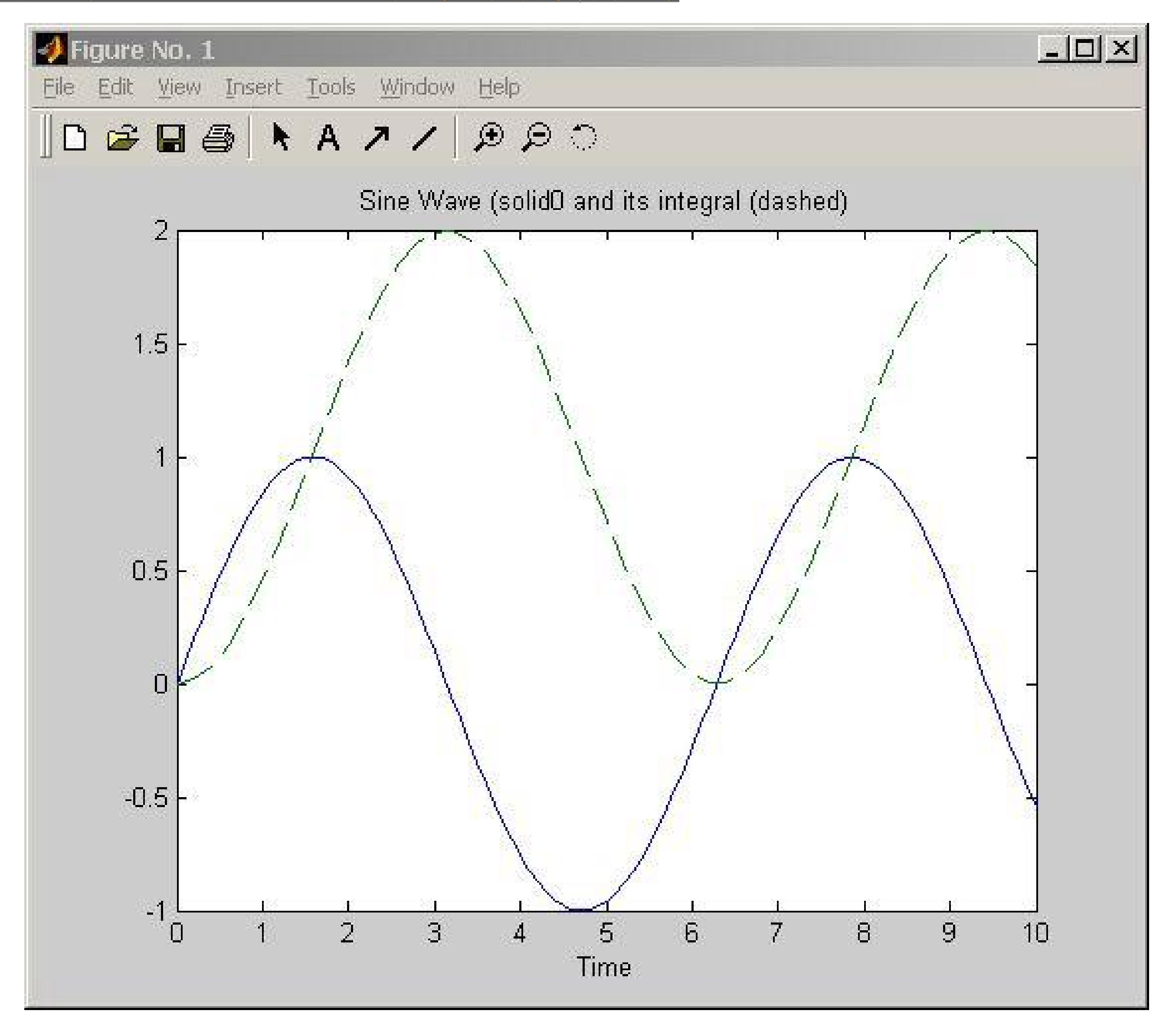


图8.3 系统原始输入与运算结果曲线

8.3 使用命令行方式进行动态系统仿真

8.3.1 使用sim命令进行动态系统仿真

1. 使用语法

```
[t,x,y] = sim(model, timespan, options, ut)
[t,x,y1, y2, ..., yn] = sim(model, timespan, options, ut);
```

以上是完整的语法格式，实际使用中可以省略其中的某些参数设置而采用默认参数进行仿真。

2. 功能描述

对指定的系统模型按照给定的仿真参数与模型参数进行系统仿真。仿真所使用的参数包括所有使用仿真参数对话框的设置、MATLAB工作空间的输入输出选项卡中的设置以及采用命令行方式设置的仿真参数与系统模块参数。

除参数'model'外，其它的仿真参数设置均可以取值为空矩阵，此时sim命令对没有设置的仿真参数使用默认的参数值进行仿真。默认的参数值由系统模型框图所决定。用户可以使用sim命令的options参数对可选参数进行设置，这样设置的仿真参数将覆盖模型默认的参数。

如果用户对连续系统进行仿真，必须设置合适的仿真求解器，因为默认的仿真求解器为变步长离散求解器（Variable Step Discrete Solver）。可以使用simset命令进行设置（后面将简单介绍simset命令的使用）。

3. 参数说明

(1) model：需要进行仿真的系统模型框图名称。

(2) timespan：系统仿真时间范围（起始时间至终止时间），可以为如下的形式：

① tFinal：设置仿真终止时间。仿真起始时间默认为0。

② [tStart　tFinal]：设置仿真起始时间（tStart）与终止时间（tFinal）。

③ [tStart OutputTimes tFinal]：设置仿真起始时间（tStart）与终止时间（tFinal），并且设置仿真返回的时间向量[tStart OutputTimes tFinal]，其中tStart、OutputTimes、tFinal必须按照升序排列。

(3) options：由simset命令所设置的除仿真时间外的仿真参数（为一结构体变量）。

(4) ut：表示系统模型顶层的外部可选输入。ut可以是MATLAB函数。可以使用多个外部输入ut1、ut2…。

(5) t：返回系统仿真时间向量。

(6) x：返回系统仿真状态变量矩阵。首先是连续状态，然后是离散状态。

(7) y：返回系统仿真的输出矩阵。按照顶层输出Outport模块的顺序输出，如果输出信号为向量输出，则输出信号具有与此向量相同的维数。

(8) y1, ... ,yn：返回多个系统仿真的输出。

8.3.2 举例之一：简单仿真

在8.2节中以动态系统command_in_out为例，对Simulink与MATLAB接口设计作了简单的回顾与复习。Simulink系统模型的输入信号为MATLAB工作空间中的变量，同时仿真结果也被输出到MATLAB工作空间中。但在进行系统仿真时，并没有使用命令行方式。实际上用户也可以使用命令行方式对此系统模型进行仿真，在仿真之前，首先使用仿真参数设置对话框设置参数。然后在MATLAB命令窗口中键入如下命令进行系统模型的仿真，并绘制此系统的输入信号与仿真结果：

```
>> t=0:0.1:10;
>> u=sin(t);
>> sim_input=[t',u'];
>> [tout, x, yout]=sim('command_in_out');
% 使用命令行sim进行系统仿真，仿真参数采用与8.2节中
  相同的仿真参数
>> plot(t,u,tout,yout,'--')
```

在对动态系统command_in_out进行仿真时，必须先打开此系统模型并且设置合适的仿真参数。sim命令的参数为模型的名称（由不带扩展名.mdl的系统模型文件名构成）。为突出sim命令的作用，这里仍使用Simulink的仿真参数设置对话框对仿真参数设置，至于使用simset命令设置仿真参数将在后面进行介绍。使用sim命令进行系统仿真前后MATLAB工作空间中的变量列表如图8.4所示。

图 8.4 中 左 侧 为 使 用 sim 命 令 进 行 仿 真 之 前 的 MATLAB工作空间变量列表（其中sim_input为系统模型的输入信号、t为输入信号的时间范围、u为输入信号的取值）；右侧为使用sim命令进行仿真之后的工作空间变量列表（其中tout为输出时间变量、x为输出系统状态变量、yout为系统运算结果；如果在系统模型的顶层没有Outport模块，则输出yout为空）。从图中可以明显看出，动态系统的仿真结果（包括输出时间向量、系统状态与运算结果）被输出到了MATLAB工作空间中。此外，由plot所绘制的图形也与8.2节中的绘制结果完全一致，这里不再给出。

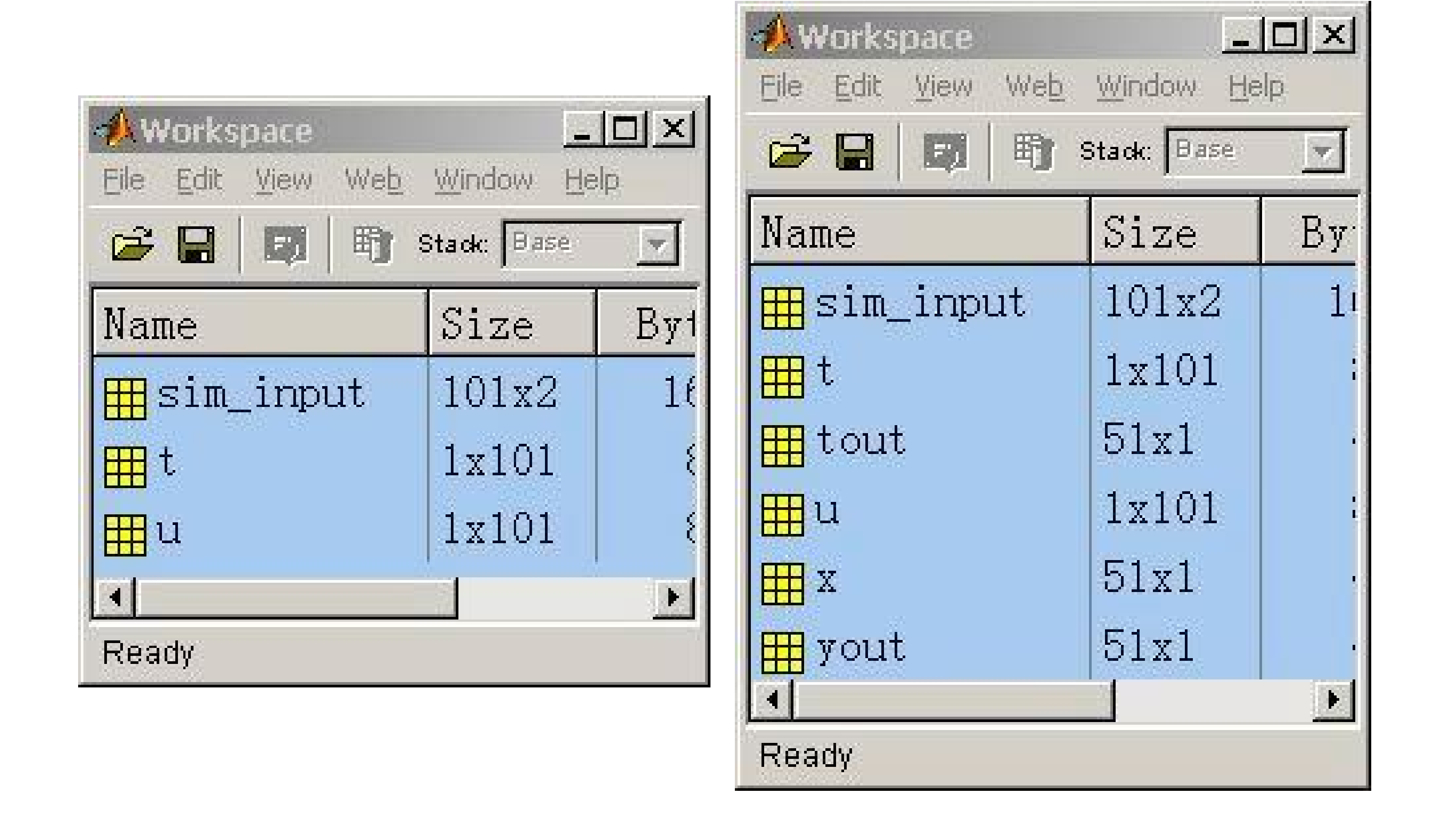


图8.4　系统仿真前后MATLAB工作空间变量列表

8.3.3举例之二：仿真时间设置

在前面已经对sim命令中的仿真时间参数timespan设置做了介绍。timespan具有三种使用形式，根据不同动态系统仿真的不同要求，用户可以选择使用如下所示的不同形式进行系统仿真：

```
>>[t,x,y]=sim(model,tFinal)
>>[t,x,y]=sim(model,[tStart tFinal])
>>[t,x,y]=sim(model,[tStart outputTimes tFinal])
```

仍以前面的动态系统command_in_out为例说明使用命令行方式时如何对系统仿真时间进行设置。首先打开系统模型command_in_out，然后仍然使用Simulink仿真参数设置对话框对系统的仿真参数进行设置，其设置与8.2节中所使用的参数完全一致。为了使用户对使用命令行方式设置系统仿真时间范围有一个直观的了解，这里使用四组不同的仿真时间对此系统进行仿真，并绘制系统输入信号与系统运算结果关系图以作比较。在MATLAB命令窗口中运行如下的命令：

```
>> t=0:0.1:10;
>> u=sin(t);
>> sim_input=[t',u'];
>> [tout1, x1, yout1]=sim('command_in_out'，5);            % 系统仿真时间范围为0至5 s，输出时间向量tout1由Simulink的求解器步长变化决定
>> [tout2, x2, yout2]=sim('command_in_out'，[1 8]);
% 系统仿真时间范围为1至8 s，输出时间向量tout1同样由Simulink的求解器步长变化决定
>> [tout3, x3, yout3]=sim('command_in_out'，1:8);
```

% 系统仿真时间范围为1至8 s，并且每隔1 s输出一次，即输出时间变量为

% [1 2 3 4 5 6 7 8]

>> [tout4, x4, yout4]=sim('command_in_out'，1:0.2:8);

% 系统仿真时间范围为1至8 s，并且每隔0.2 s输出一次，即输出时间变量为

% [1 1.2 % 1.4 … 7.6 7.8 8]

>> subplot(2,2,1)

>> plot(t,u,tout1,yout1,'.');

>> subplot(2,2,2)

```
>> plot(t,u,tout2,yout2,'.');
>> subplot(2,2,3)
>> plot(t,u,tout3,yout3,'.');
>> subplot(2,2,4)
>> plot(t,u,tout4,yout4,'.');
% 在同一幅图中绘制系统command_in_out的输入信号以
  及在不同仿真时间范围上的系统运算结
% 果，其结果如图8.5所示
```

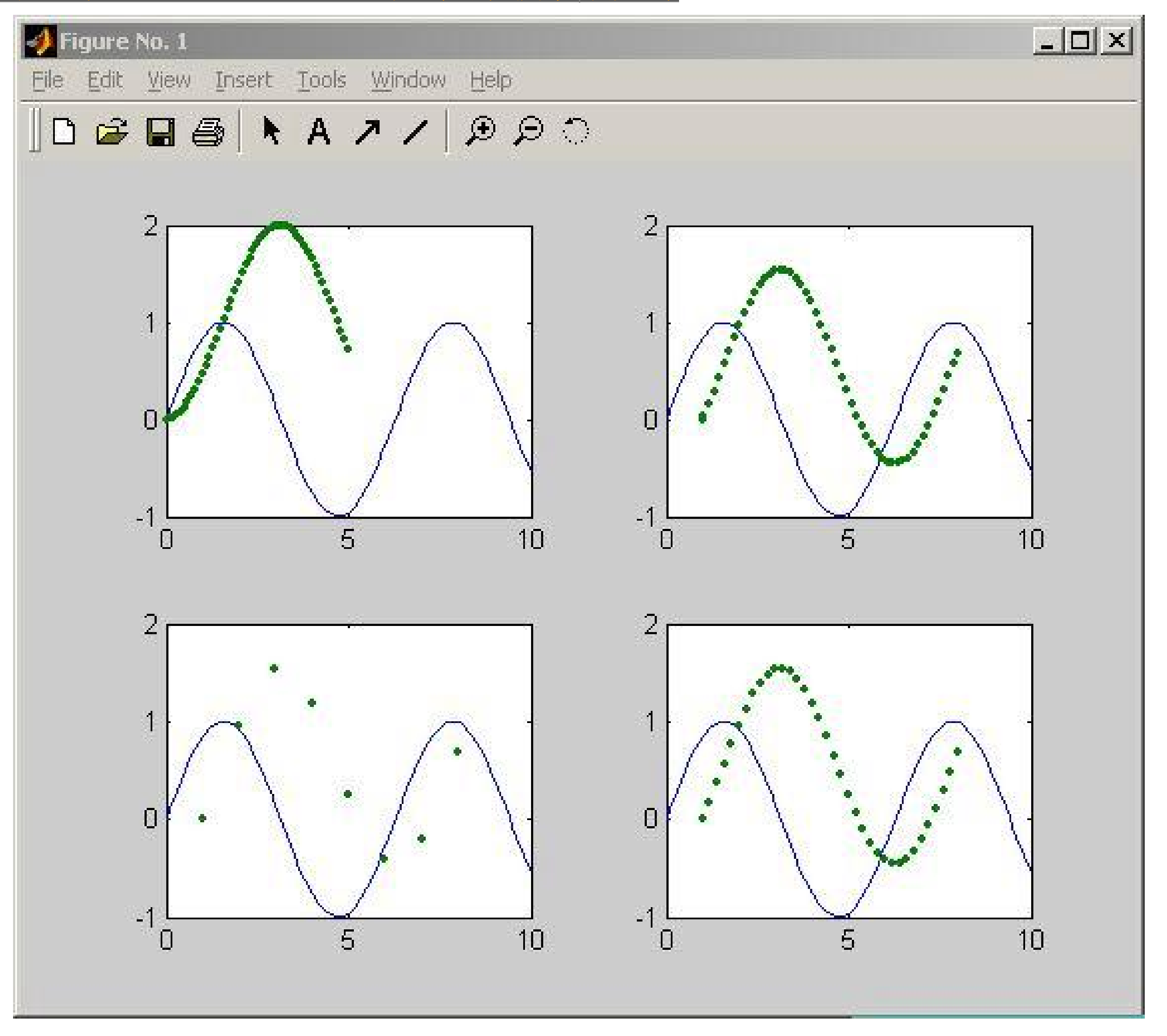
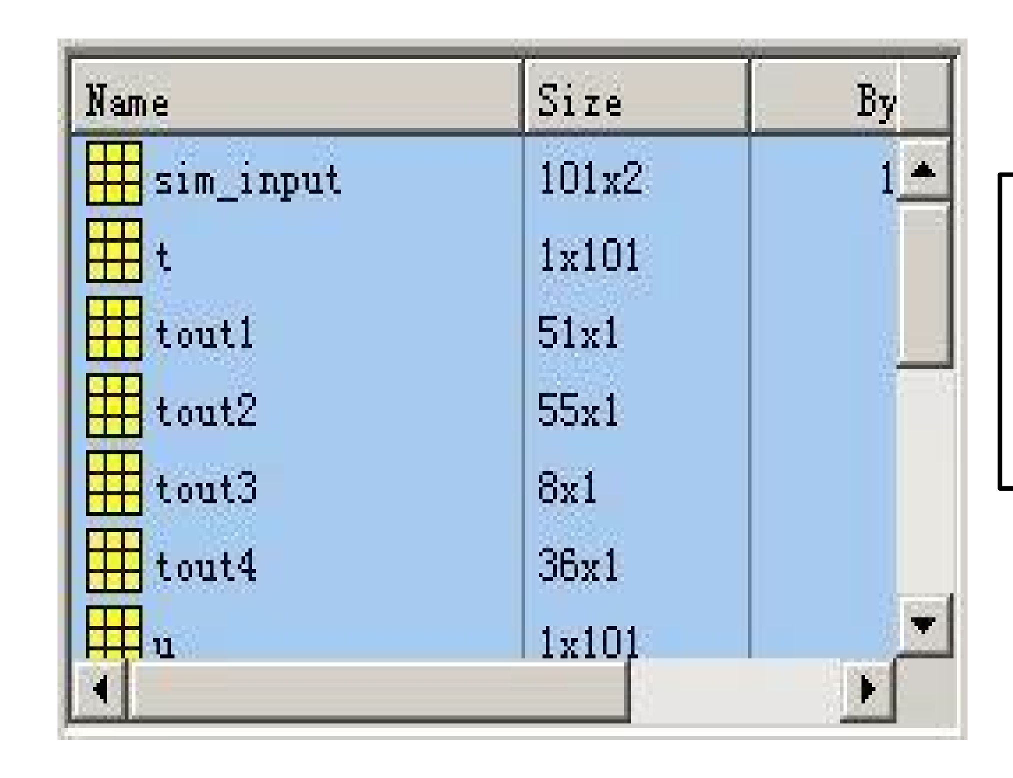


图8.5 在不同仿真时间设置下的系统仿真结果比较



从MATLAB工作空间中查看不同仿真时间设置下的系统输出结果

图8.6 系统仿真返回的时间向量输出

8.3.4 举例之三：外部输入变量设置

前面对动态系统command_in_out进行仿真时，通过设置Simulink仿真参数设置对话框中Workspace I/O中的外部变量输入，以使系统在仿真过程中从MATLAB的工作空间中获取输入信号sim_input。除了使用这种方法从MATLAB工作空间中获得系统输入信号之外，用户还可以通过使用sim命令中的ut参数来设置系统的外部输入信号。下面介绍如何使用ut参数设置外部输入信号。

1. ut参数的生成

用户可以使用命令 [t,x,y]=sim(model, timespan, options, ut) 对动态系统进行仿真并且从MATLAB工作空间中输入变量。其中ut为一个具有两列的矩阵，第一列表示外部输入信号的时刻，第二列表示与给定时刻相应的信号取值。使用矩阵ut能够为系统模型最顶层的Inport模块提供外部输入，并将自动覆盖Simulink仿真参数设置对话框中Workspace I/O中的设置。此外，当输入信号中存在着陡沿边缘时，必须在同一时刻处定义不同的信号取值。例如，对于图8.7所示的一个类似于方波的信号。

产生此输入信号的MATLAB命令为

```
>>ut=[0 1;10 1;10 -1;20 -1;20 1;30 1;30 -1;40 -1;40 1;50 1]
```

2. 应用举例

仍以动态系统command_in_out为例说明如何使用sim命令的参数ut从MATLAB工作空间中获得输入信号。在MATLAB命令窗口中键入如下的命令：

```
>> t=0:0.1:10;
>> u=sin(t);
>> sim_input=[t',u'];
>> [tout1, x1, yout1]=sim('command_in_out'，10);
% 使用Simulink仿真参数对话框中的Workspace I/O从MATLAB工作空间中获得输入信号
>>u=cos(t);
```

```
>>ut=[t',u'];                % 改变系统输入信号
>>[tout2,x2,yout2]=sim('command_in_out',10,[],ut);
% 使用sim命令的ut参数获得系统输入信号，ut的使用会
  覆盖由Workspace I/O的系统输入设置，
% 这一点可以在下面的系统仿真结果图形中反映出来
>>subplot(1,2,1); plot(tout1,yout1);
>>subplot(1,2,2);plot(tout2,yout2);
%绘制系统在不同输入信号下的响应曲线，如图8.8所示
```

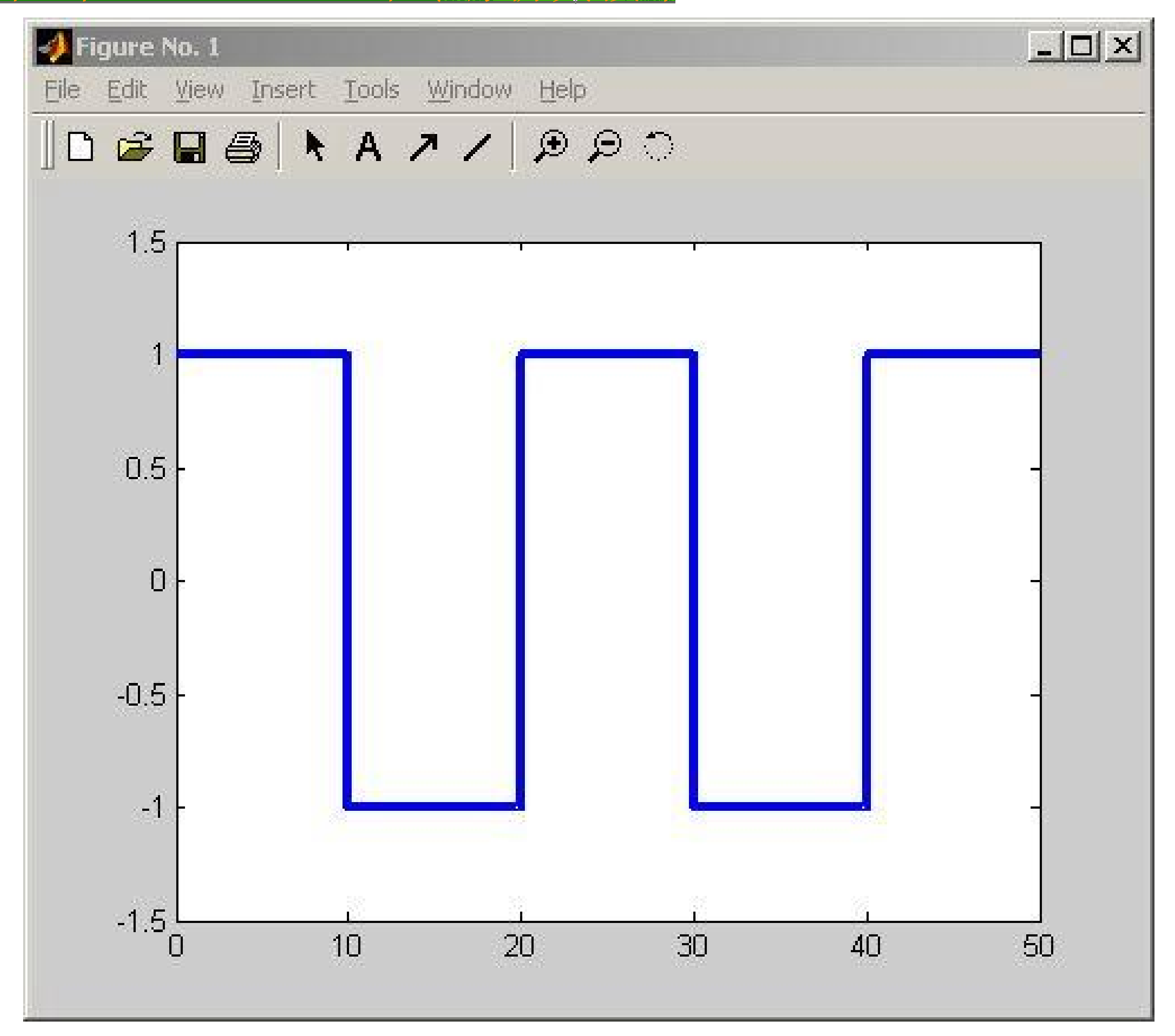

Figure No. 1
File Edit View Insert Tools Window Help
1.5
1
0.5
0
-0.5
-1
-1.5
0
10
20
30
40
50

图8.7 类方波信号

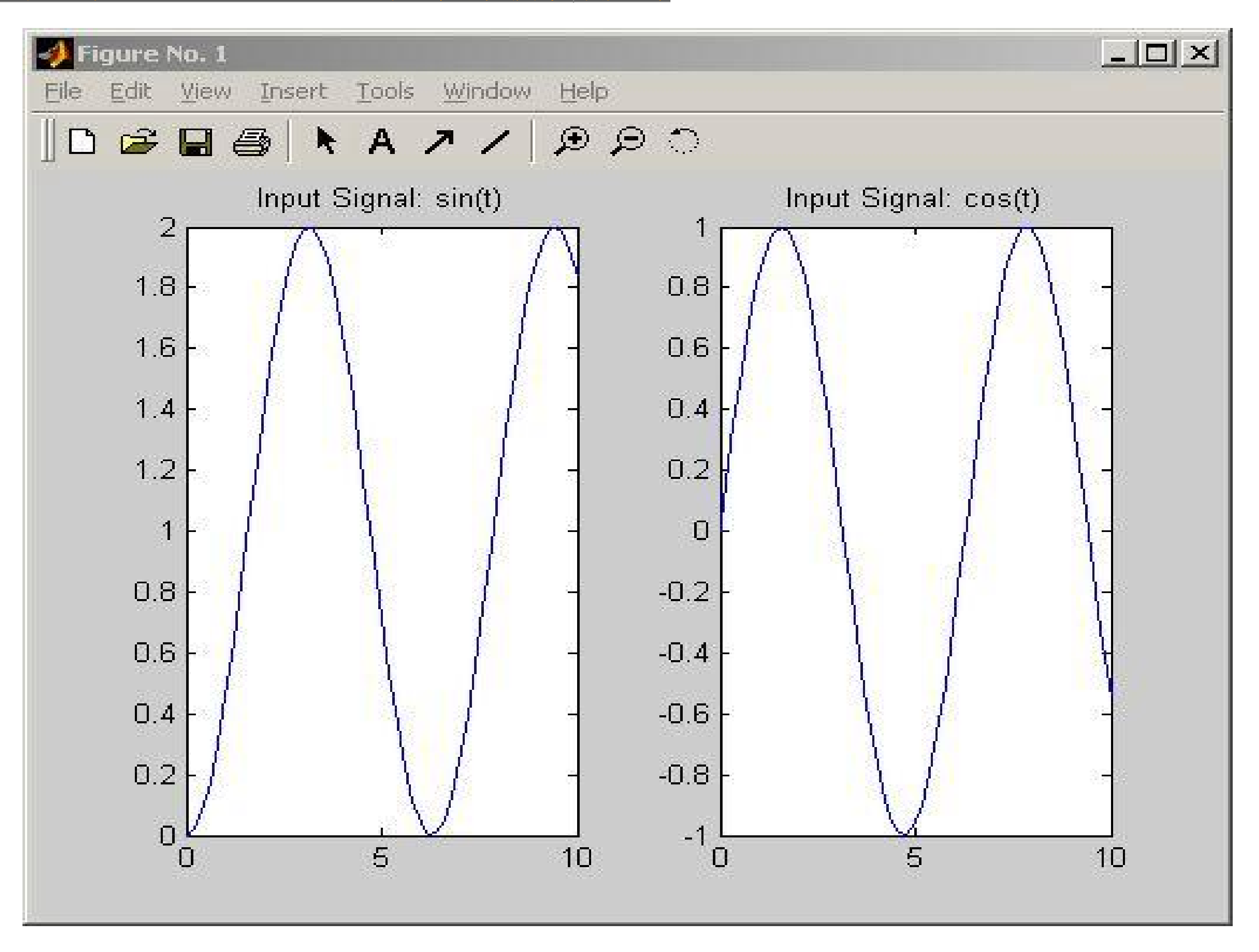


图8.8 系统在不同输入下的响应曲线

8.3.5 simset与simget命令的使用

前面介绍使用sim命令进行动态系统仿真时，除了对系统仿真时间与系统输入的设置之外，其它所有的仿真参数均是由Simulink的仿真参数设置对话框所设置的。用户也可以使用simset对系统仿真参数进行设置，然后再使用命令行方式对系统进行仿真分析，如下所示：

```
>>[t,x,y] = sim(model, timespan, options, ut);
```

其中options为仿真选项结构体变量（不包括系统仿真时间），使用simset命令可以对此结构体变量进行设置。注意，这里所谓的仿真选项指的就是Simulink仿真参数设置对话框Simulation parameters中所设置的参数。为了使用户对options系统仿真选项有一个总体的了解，首先使用simget命令获得表示系统仿真参数的结构体变量，如下所示：

```
>>options=simget('command_in_out')
%获得已经打开的系统模型command_in_out的仿真参数选项
options =
```

AbsTol : 'auto'

Debug: 'off'

Decimation : 1

DstWorkspace: 'current'

FinalStateName : ''

FixedStep : 'auto'

InitialState : []

InitialStep : 'auto'

MaxOrder : 5

SaveFormat: 'Array'

MaxDataPoints: 1000

```
MaxStep: 'auto'
MinStep: []
OutputPoints: 'all'
OutputVariables: 'ty'
Refine: 1
RelTol: 0.0010
Solver: 'ode45'
SrcWorkspace: 'base'
Trace: ''
ZeroCross: 'on'
```

1. simset命令使用介绍

1) 使用语法

```
options = simset(property, value, ...);
options = simset(old_opstruct, property, value, ...);
options = simset(old_opstruct, new_opstruct);
simset
```

2) 功能描述

使用simset命令会产生一结构体变量options，此变量中的各个数据用来设置系统仿真参数。对于没有进行设置的系统仿真参数，Simulink会使用相应的缺省值。

options=simset(property，value，...)：设置指定的仿真参数选项值。其中property为指定的仿真参数，value为指定的取值。

options=simset(old_opstruct, property, value, ...)：修改仿真参数结构体变量中已经存在的指定仿真参数选项。其中old_opstruct表示已经存在的结构体。

options=simset(old_opstruct，new_opstruct)：合并已经存在的两个仿真参数结构体变量，并且使用new_opstruct中的域值覆盖old_opstruct中具有相同域名的域值。

simset：显示所有的仿真参数选项及其可能的取值。

3) 仿真参数选项介绍

使用simset命令可以设置除仿真时间外的所有系统仿真 参数选项，下面对最常用的仿真参数选项及其取值作一个简单的介绍。

(1) AbsTol：取值为一标量，缺省值为1e–6，表示绝对误差限。仅用于变步长求解器。

(2) Decimation：取值为一正整数，缺省值为1，表示系统仿真结果返回数据点的间隔（值为1，表示每一仿真结果数据均返回到相应的变量中；值为2，表示仿真结果每隔一个数据点返回到相应的变量中，依次类推）。

(3) FixedStep：取值为正值标量，表示定步长求解器的步长。如果对离散系统进行仿真，其缺省取值为离散系统的采样时间；如果对连续系统进行仿真，其缺省取值为仿真时间范围的1/50。

(4) InitialState：取值为一向量，缺省值为空向量，表示系统的初始状态。如果系统中同时存在连续状态与离散状态，则此向量中先是连续状态的初始值，然后是离散状态的初始值。初始状态的设置会覆盖系统模型中所默认的状态初始值。

(5) InitialStep：取值为一正标量，缺省值为auto，表示系统仿真时的初始步长（估计值），仅用于变步长求解器；在仿真时求解器首先使用估计的步长，缺省情况下由求解器决定初始仿真步长。

(6) MaxStep：取值为一正值标量，缺省值为auto，表示系统仿真的最大步长。仅用于变步长求解器，缺省情况下最大仿真步长为仿真时间范围的1/50。

(7) RelTol：取值为一正值标量，缺省值为1e–3，表示相对误差限。仅用于变步长求解器。

(8) Solver：表示Simulink仿真求解器，其取值为如下由“|”隔开的字符串：VariableStepDiscrete | ode45 | ode23 | ode113 | ode15s | ode23s | FixedStepDiscrete | ode5 | ode4 | ode3 | ode2 | ode1。在缺省值为变步长连续求解器，运算方法为ode45（Dormand-Prince）。

(9) ZeroCross：表示仿真过程中的过零检测，取值为on或off，缺省值为on。仅用于变步长求解器。on表示对系统的模块进行过零检测，off表示不进行过零检测。

4) 应用举例

```
>>myoptions＝simset(‘ZeroCross’,’off’);
% 关闭系统仿真过零检测
>>[tout,x,yout]=sim(‘command_in_out’, 10, myoptions);
% 使用myoptions仿真参数选项进行系统仿真
```

2. simget命令的使用介绍

1) 使用语法

```
struct = simget(model)
value = simget(model, property)
value = simget(OptionStructure, property)
```

2) 功能描述

simget命令用来获得指定系统模型的仿真参数设置。

struct = simget(model)：获得指定系统模型model的所有仿真参数设置结构体变量。

value = simget(model, property)：获得指定系统模型model中指定仿真参数property的取值。

value = simget(OptionStructure, property)：获得系统仿真长参数选项中指定的仿真参数的取值。

8.3.6 simplot命令的使用

在观测动态系统仿真结果时，Sinks模块库中的Scope模块是最常用的模块之一。它可以使用户在类似示波器的图形界面中观测系统仿真结果的输出，而非使用诸如plot绘制系统的输出。使用Scope最重要的优点是可以通过对Scope模块的操作，对系统输出进行方便的观测，这一点是plot等命令所不及的。在使用命令行方式对动态系统进行仿真时，可以使用simplot命令绘制与使用与Scope模块相类似的图形。下面介绍simplot命令的功能与使用。

1. 使用语法

```
simplot(data);
simplot(time, data);
```

2. 功能描述

simplot命令用来输出动态系统的仿真结果，且其输出图形与Scope模块的输出类似。

3. 参数说明

data：动态系统仿真结果的输出数据（一般由Outport模块、To Workspace模块等产生的输出）。其数据类型可以为矩阵、向量或是结构体等。time：动态系统仿真结果的输出时间向量。当系统输出数据为带有时间向量的结构体变量时，此参数被忽略。

4. 举例

仍以前面的command_in_out为例说明simplot的使用（设系统仿真结果已经存在）。在MATLAB命令窗口中键入如下命令：

```
>>simplot(tout2,yout2)                 % 使用simplot绘制系统仿真结果，如图8.9所示
```

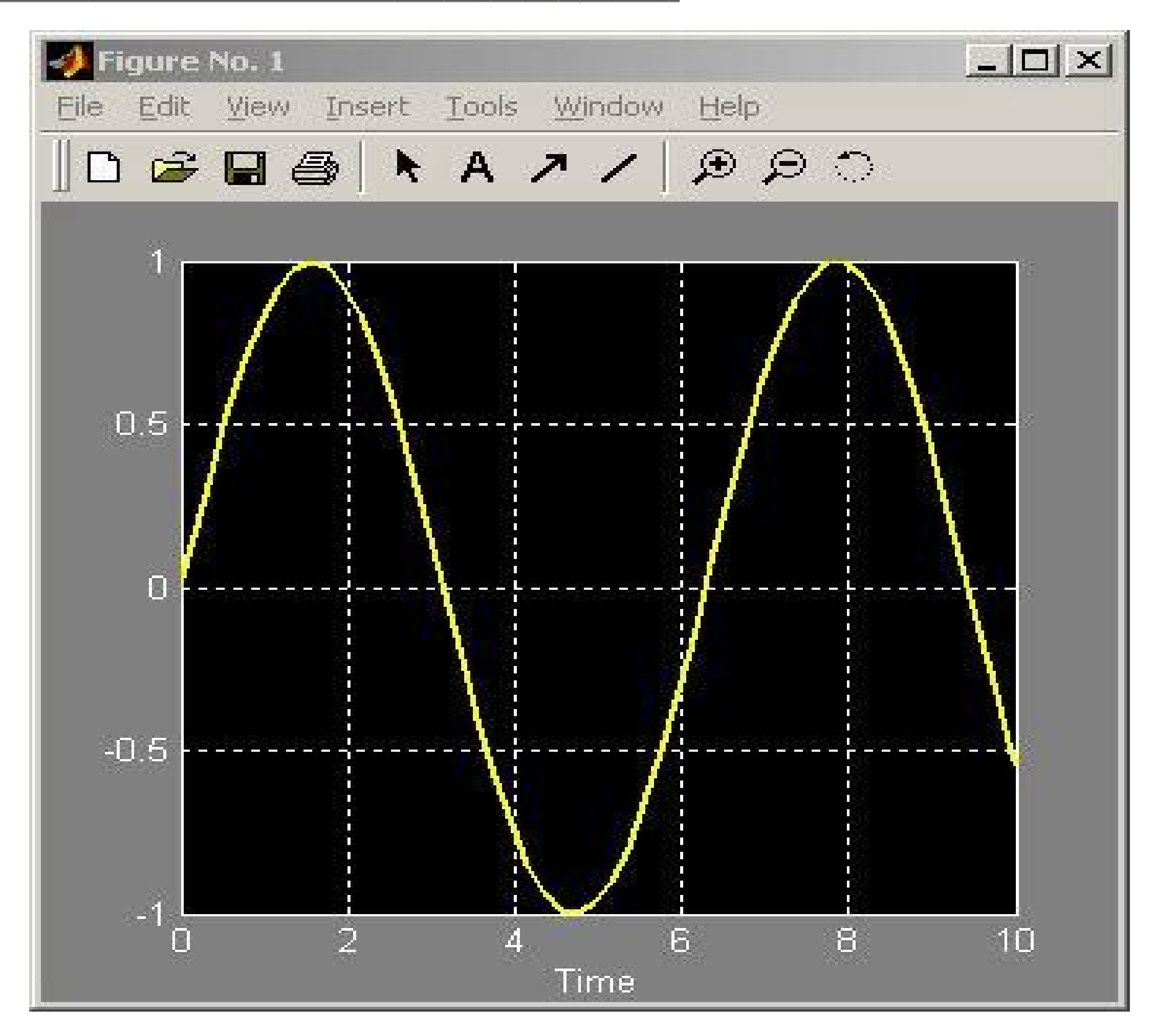


图8.9 使用simplot命令绘制系统仿真结果

8.4 使用MATLAB脚本分析动态系统

8.4.1 蹦极跳的安全性分析

在第5章中曾给出蹦极跳连续系统模型：

$$m\ddot{x} = mg + bx - a_1\dot{x} - a_2\left|\dot{x}\right|\dot{x}$$

其中为蹦极者的质量，为重力加速度，为物体的位置，与表示空气的阻力，而

$$b(x) = \begin{cases} -kx, & x > 0 \\ 0, & x \le 0 \end{cases}$$

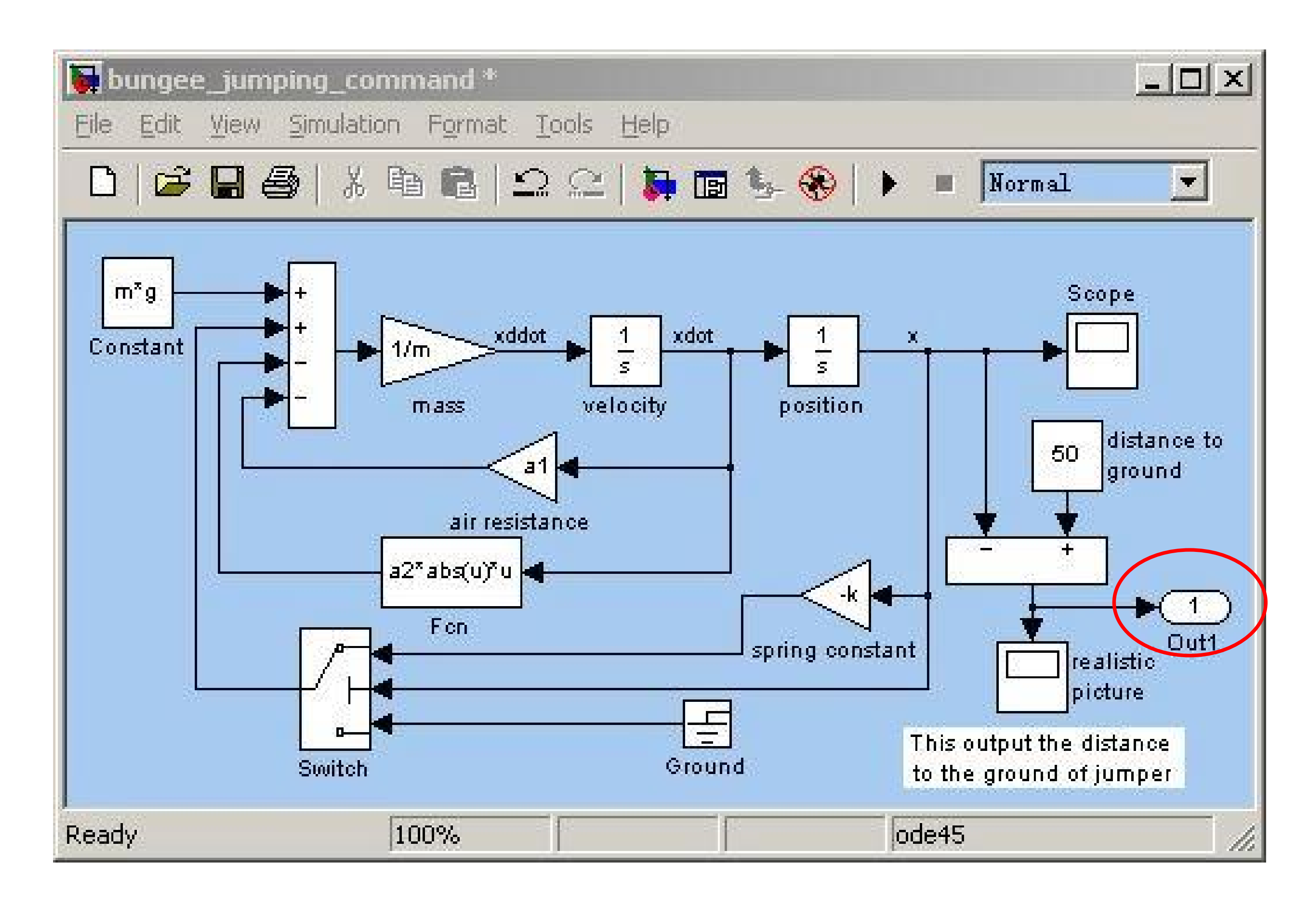


图8.10 蹦极跳系统模型框图

下面使用命令行方式对此系统在不同的弹性常数下进行仿真分析，以求出符合安全要求的弹性绳索的最小弹性常数。编写MATLAB脚本文件bungee_jumping_cmd.m，求取最小弹性常数，程序如下所示：

```
m=70;
g=10;
a1=1;                    % 使用MATLAB工作空间中变量设置
a2=1;                    % 系统模型中模块的参数
for k=20:50                   % 在一定的范围内试验弹性常数
```

```
[t,x,y]=sim('bungee_jumping_command',[0 100]);
  %使用不同的弹性常数进行系统仿真
 if min(y)>0              % 如果仿真结果输出数据的最小值大于0，
    break;      % 则说明此弹性常数符合安全要求，跳出循环
  end
end
```

```
disp(['The minimum safe k is: ',num2str(k)])          % 显示最小安全弹性常数
dis=min(y);                        % 求取蹦极者与地面之间的最小距离
disp(['The minimum distance between jumper and ground is: ', num2str(dis)])
simplot(t,y)                       % 绘制最小安全弹性常数下系统的仿真结果
```

在运行此MATLAB脚本文件求取最小安全弹性常数之前，必须正确设置蹦极者的初始位置与初始速度（用户可以参考第5章中相应的参数设置，这里不再赘述）。然后再运行此脚本文件，以求取最小安全弹性常数和在此弹性常数下蹦极者与地面之间的最小距离，并且显示在此弹性常数下的蹦极跳系统动态过程。其中最小安全弹性常数、蹦极者与地面之间的最小距离在MATLAB命令窗口中显示如下：

```
>> The minimum safe k is: 27
The minimum distance between jumper and ground is:
    0.87934
```

在最小安全弹性常数为27的情况下，体重为70 kg的蹦极者与地面之间的最小距离不足1 m。图8.11所示为此系统的动态过程。

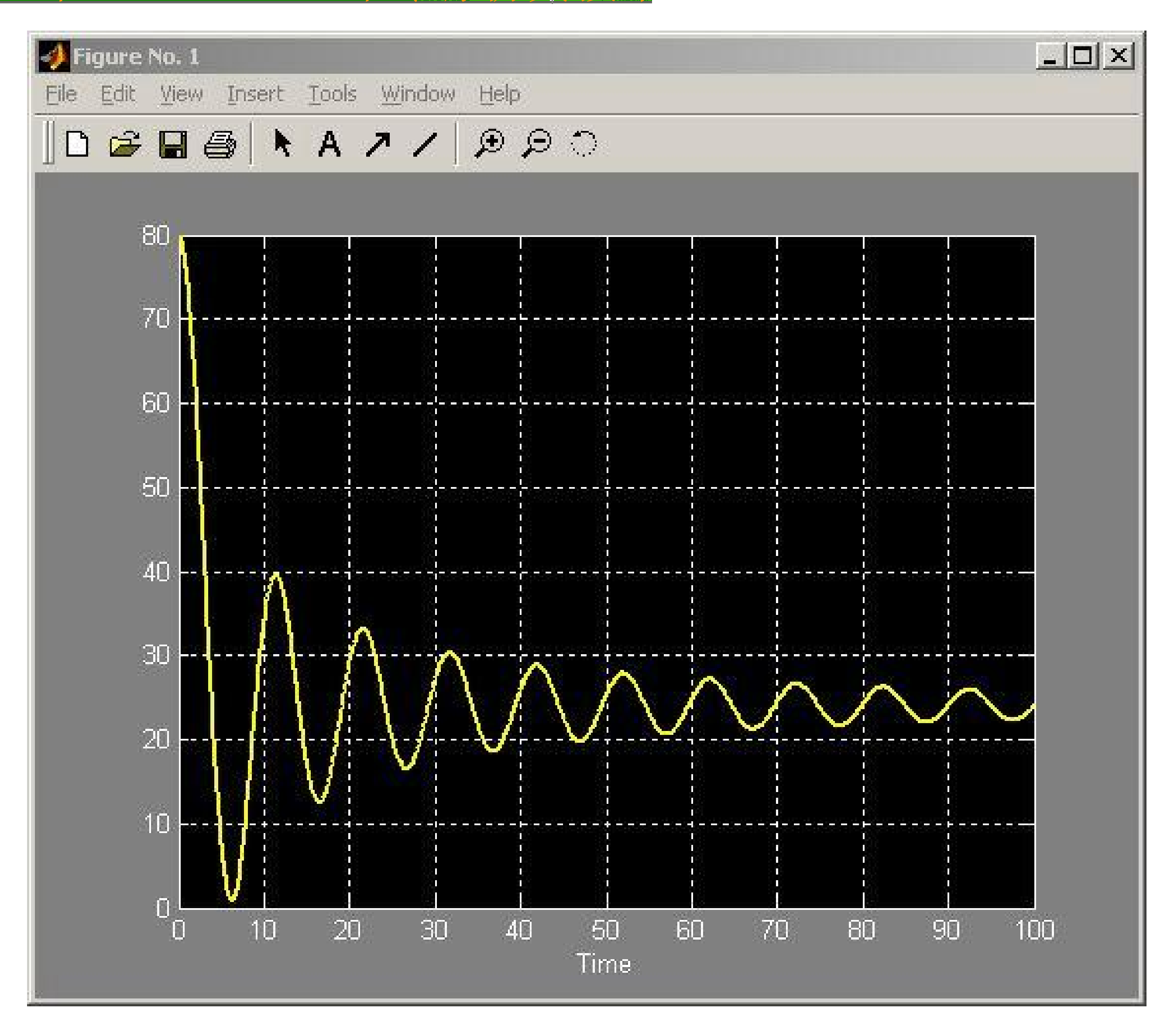


图8.11　最小安全弹性常数下的系统动态过程

8.4.2行驶控制系统中控制器的调节

第5章中的行驶控制系统给出了系统综合设计仿真的一个简单实例。汽车行驶控制系统的基本目的是控制汽车的速度变化，使汽车的速度在合适的时间之内加速到指定的速度。由第5章中的介绍可知，汽车行驶控制系统由如下三部分所构成：

(1) 速度操纵机构的位置变换器。

(2) 行驶控制器。

(3) 汽车动力系统。

其中最为重要的部分为行驶控制器，行驶控制器是一个典型的PID反馈控制器。现要求使用命令行方式对行驶控制系统中的行驶控制器中比例调节的性能进行定性的分析。已知行驶控制器中PID控制器的I（积分）、D（微分）参数如下取值分别为I=0.01、D=1。P（比例）的取值由MATLAB脚本文件所决定（以分析不同P值下行驶控制器的性能）。为了对行驶控制器中比例调节的性能进行定性的分析，需要对第5章的行驶控制系统模型做一些改变，如下所示：

(1) 将行驶控制器子系统中的比例增益的取值改为p。

(2) 在系统模型的最顶层加入一个Outport模块输出仿真结果。

图8.12所示为修改后的汽车行驶控制系统的系统模型框图。

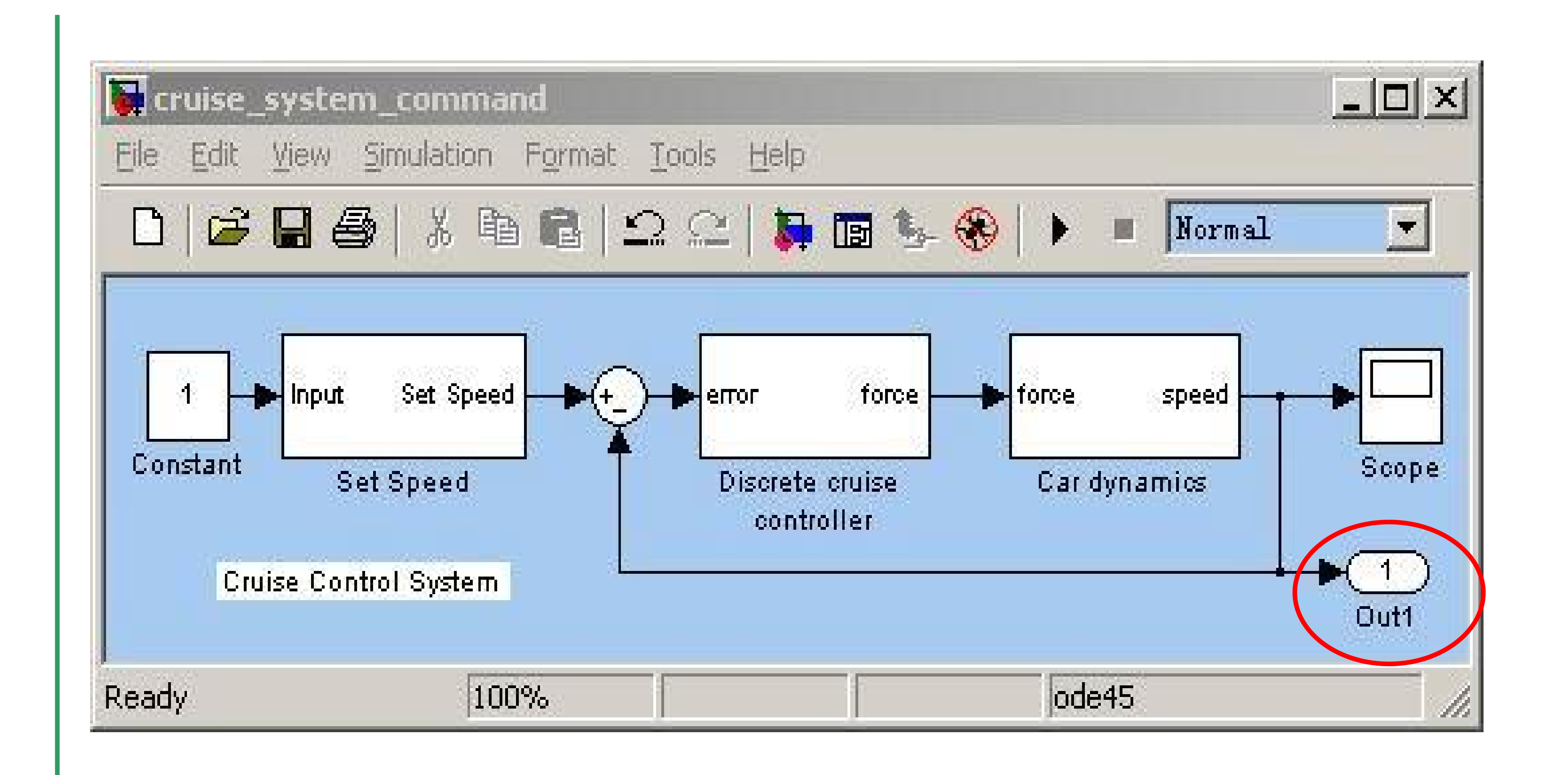


图8.12 修改后的汽车行驶控制系统模型

然后编写MATLAB脚本文件cruise_system_cmd.m对行驶控制系统在不同的比例调节器取值下进行仿真，并绘制出不同取值下系统仿真结果以对比例调节性能进行分析。其要求如下：

(1) 编写一个for循环改变p的值，取值范围为0到25，间隔为5。

(2) 在同一幅图中作出不同p值下系统的仿真结果以对比例调节性能进行分析。

编写好的cruise_system_cmd.m脚本文件如下所示：

```
for p=0:5:25          % 设置不同的比例调节器取值
[t,x,y]=sim('cruise_system_command');    % 对系统进行仿真
subplot(3,2,p/5+1)% 绘制在此取值下系统仿真结果，其
plot(t,y)                       % 结果如图8.13所示
ylabel(['p=',num2str(p)])
end
```

从系统仿真的结果中可以明显看出比例调节器取值对汽 车行驶控制系统性能的影响：增加比例调节器的取值可以有效的改善行驶控制系统的动态性能。这是因为，对于行驶控制系统而言，其速度变化越平稳越好（但并非变化缓慢）。从图中可以看出，对于取值较大的比例调节器，汽车速度的过渡时间较小，而且变化平稳（仿真结果曲线无振荡，光滑）。

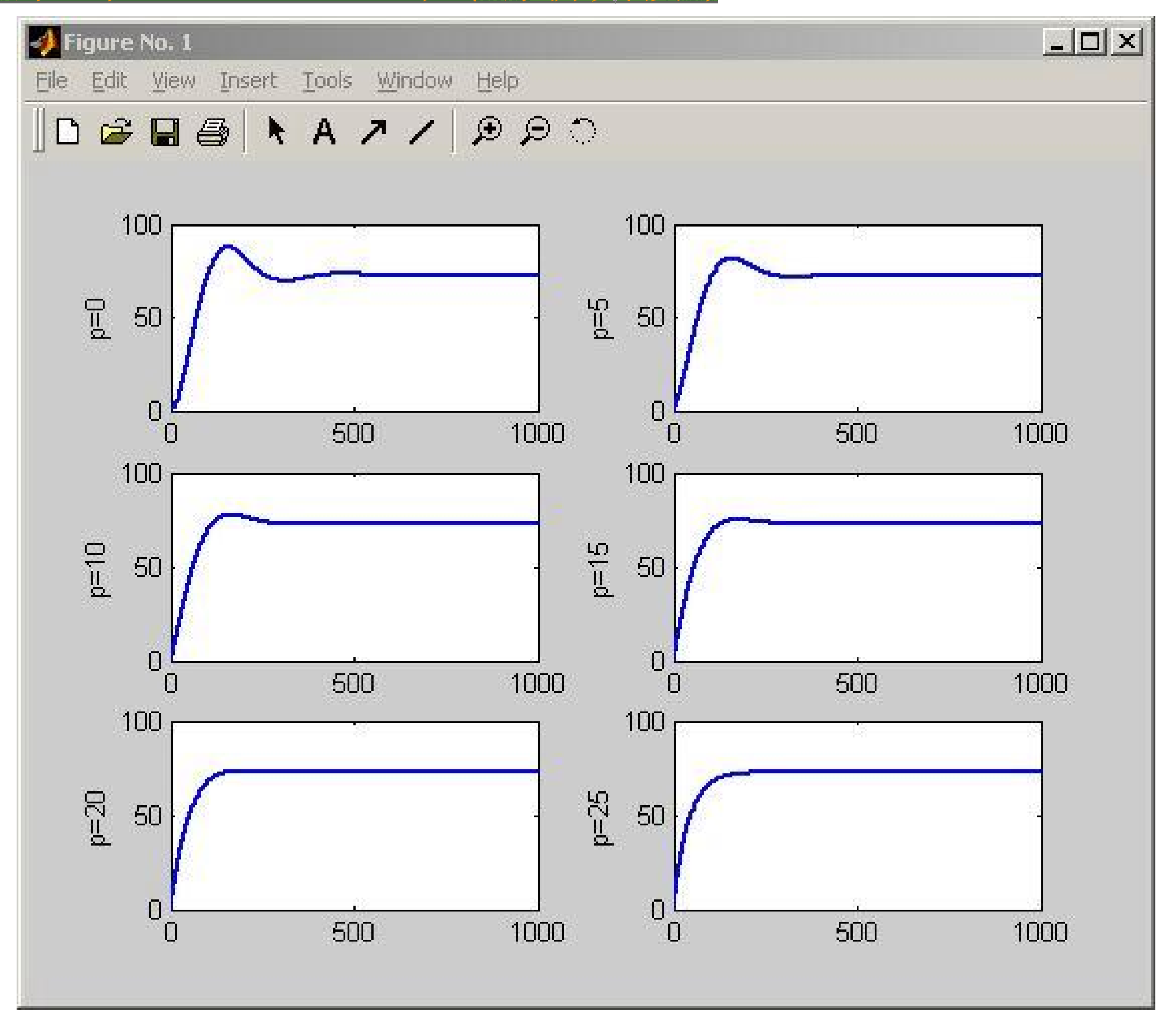


图8.13 不同比例调节器取值下的系统响应

8.5 其 它 内 容

8.5.1 系统状态的确定

当使用命令行方式进行动态系统仿真时，经常需要对系统中的状态变量进行分析。当动态系统的系统模型中存在多个状态变量时，对这些状态变量进行分析（尤其是对输出的多个状态变量中的某一个状态进行单独分析），需要清楚地知道状态变量矩阵中各个状态变量的顺序。这时，用户可以使用下面的命令获得系统模型中状态输出的顺序。

```
>> [sizes, x0, xord]=modelname
```

例如，对于蹦极跳系统，如果需要对蹦极者在某个时刻的位置position与速度velocity进行分析，首先需要设置Simulink的仿真参数设置对话框中Workspace I/O选项卡以输出系统中的状态变量，如图8.14所示。

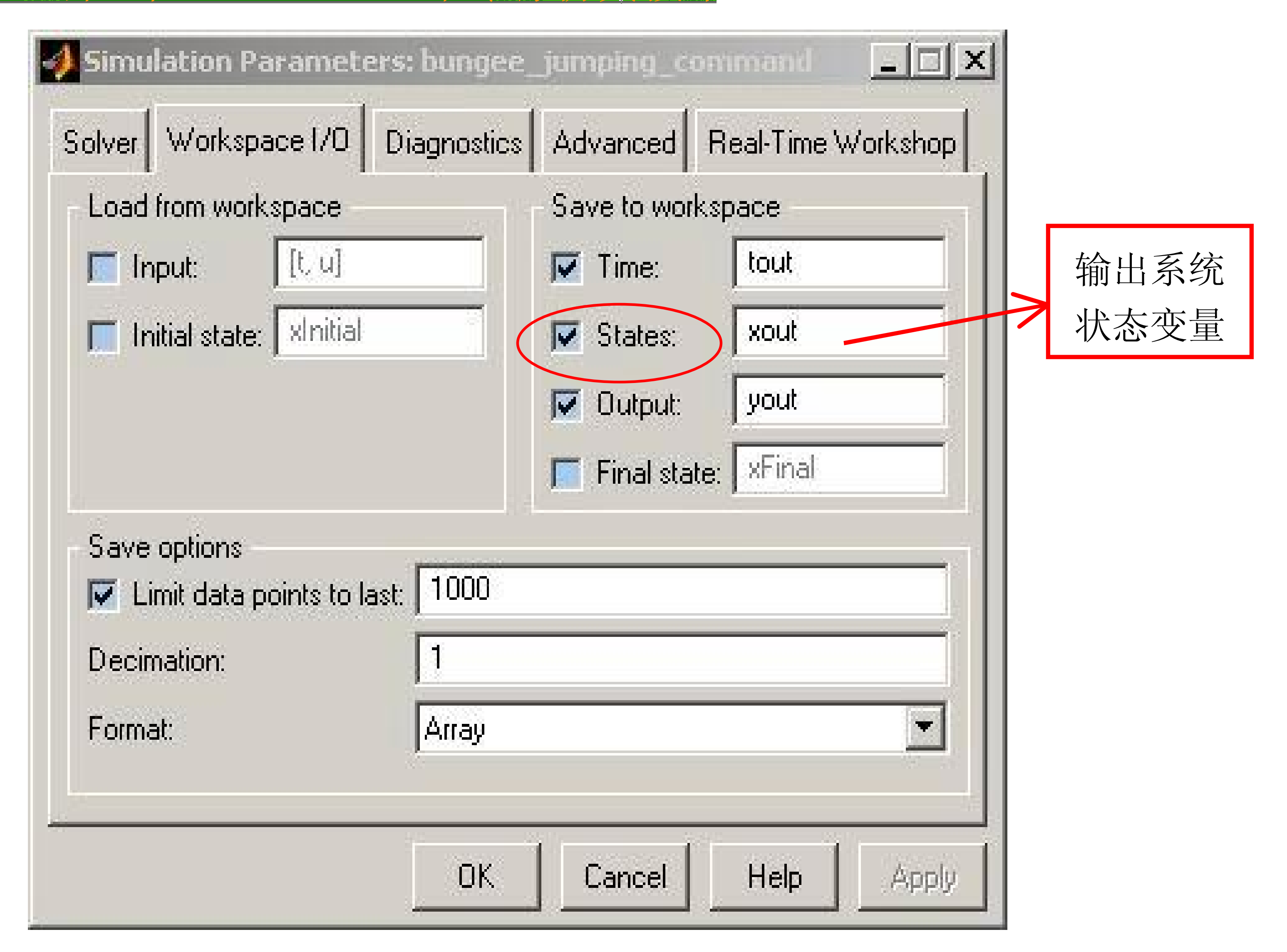


图8.14　设置Workspace I/O选项卡以输出系统状态

从中可以知道蹦极跳系统的诸多信息，如系统的输入输出变量的数目（外部输入数目为0、输出1个变量）、系统中产生状态变量的模块及其输出顺序（第一个状态为积分块position的输出，即蹦极者的位置；第二个状态为积分块velocity的输出，即蹦极者的速度）、各状态变量的初始取值（积分块position的初始取值为–30，积分块velocity的初始取值为0）等等信息。此时使用MATLAB的Array Editor观测输出变量xout，其中第一列与第二列分别为蹦极者的位置与速度，如图8.15所示。

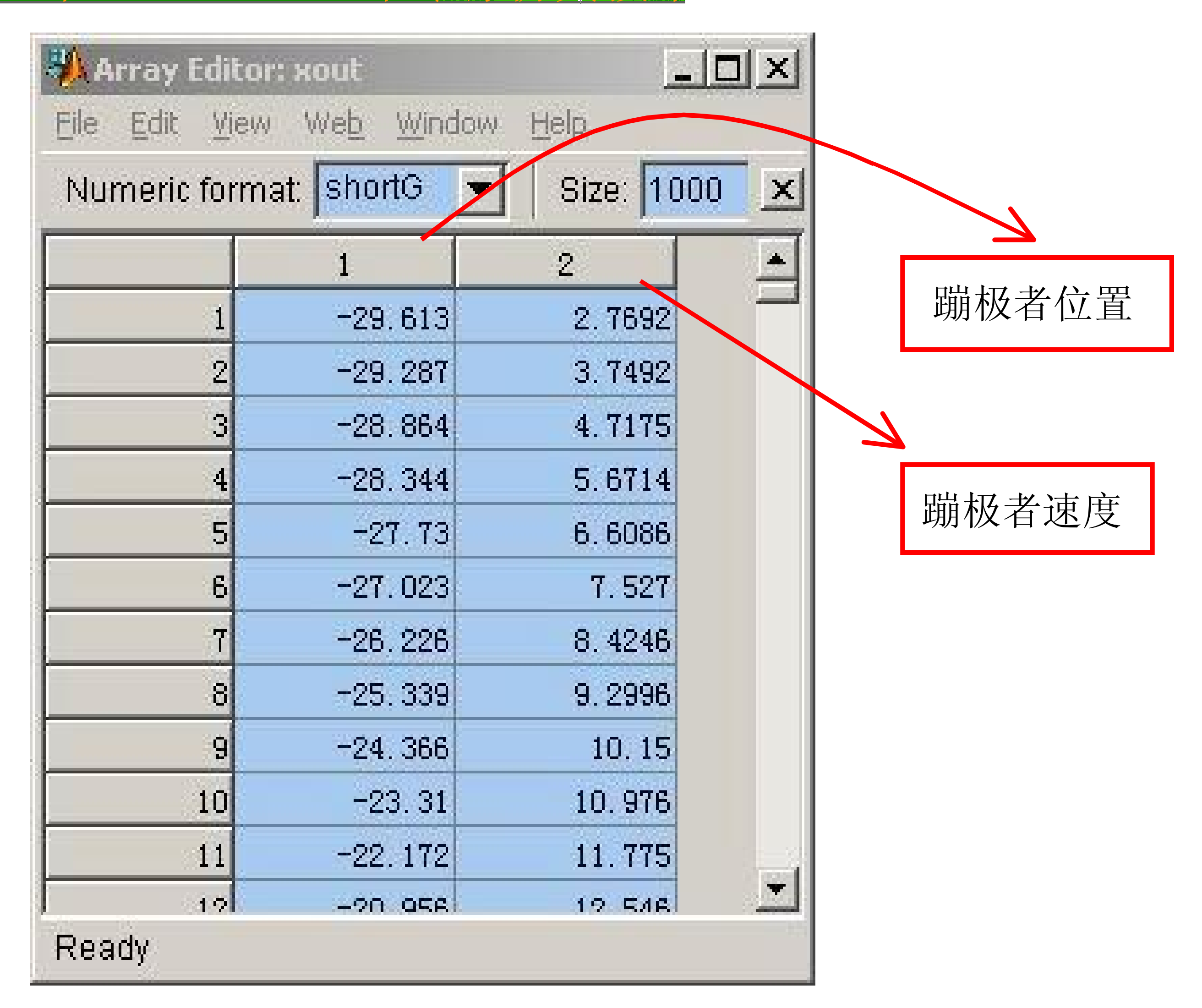


图8.15 蹦极跳者位置与速度

8.5.2 系统平衡点的确定

在大多数的系统设计中，设计者都需要对所设计的系统进行稳定性分析，因为绝大多数的系统在运行之中，都需要按照某种方式收敛到指定的平衡点处。这里所谓的平衡点一般指的是系统的稳定工作点，此时系统中所有的状态变量的导数均为0，系统处于稳定的工作状态。本书称处于平衡点状态时的系统为静止系统。

在系统设计中，最主要的设计目的之一便是使系统能够满足系统稳定的要求并在指定的平衡点处正常工作。在使用Simulink进行动态系统设计、仿真与分析时，可以使用命令trim对系统的稳定性与平衡点进行分析。下面简单介绍一下trim的使用与功能。

1. 使用语法

```
[x,u,y,dx] = trim('sys')
[x,u,y,dx] = trim('sys',x0,u0,y0)
[x,u,y,dx] = trim('sys',x0,u0,y0,ix,iu,iy)
[x,u,y,dx] = trim('sys',x0,u0,y0,ix,iu,iy,dx0,idx)
[x,u,y,dx] = trim('sys',x0,u0,y0,ix,iu,iy,dx0,idx,options)
[x,u,y,dx] = trim('sys',x0,u0,y0,ix,iu,iy,dx0,idx,options,t)
[x,u,y,dx,options] = trim('sys',...)
```

2. 功能描述

根据系统的输入、初始状态（也可以说是初始的工作点），按照一定的方法求取系统中距离此工作点最近的平衡点，以及在达到平衡点时的系统输入与输出。如果系统中不存在平衡点，则trim命令会返回系统状态变量最接近0的工作点。

[x,u,y] = trim('sys')：求取距离给定初始状态x0最近的系统平衡点。

[x,u,y] = trim('sys',x0,u0,y0)：求取距离给定初始状态x0、初始输入u0与初始输出y0最近的平衡点。

[x,u,y,dx] = trim('sys',x0,u0,y0,ix,iu,iy)：求取距离给定初始值向量中某一初始值距离最近

[x,u,y,dx] = trim('sys', x0, u0, y0, ix, iu, iy, dx0, idx)：求取非平衡点，此平衡点处的系统状态为指定值。其中dx0为指定状态值向量，idx为相应的序号。

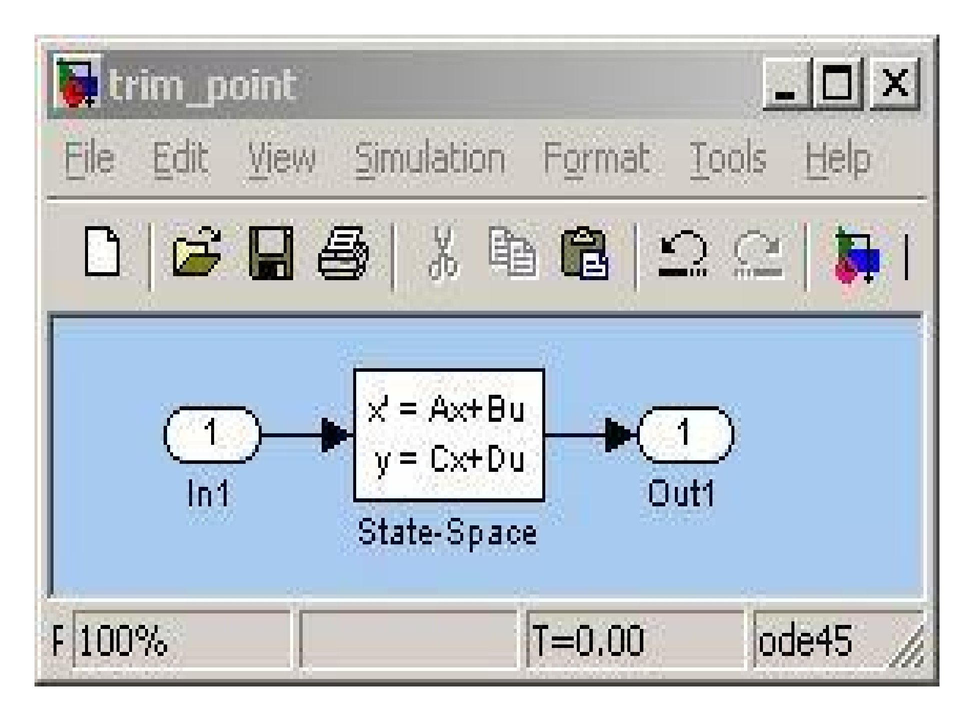
至于trim命令中的options选项，其功能是用来优化平衡点的求取，这里不再赘述。需要使用此功能的用户可以参考Simulink中trim命令的联机帮助。

3. 注意事项

在使用trim命令求取系统的平衡点时，所求取的平衡点为局部最优平衡点，而非全局最优平衡点，因此如果需要求取全局最优平衡点，需要使用多个初始状态进行搜索。这是因为对于很多系统而言，系统中平衡点的数目往往不止一个。

4. 应用举例

对于图8.16所示的系统，试求取其在初始状态x0=[1;1]以及初始输入u0=[1;1]下的平衡点。





系统模型参数

A=[−0.09	−0.01;	1	0]
B=[0	−7;	0	−2]
C=[0	2;	1	−5]
D=[−3	0;	1	0]

图8.16　由状态空间描述的线性系统模型框图

其实只需要在MATLAB的命令窗口中键入如下的命令便可以求取系统在此初始状态与初始输入下的平衡点：

```
>> x0=[1;1];
>>u0=[1;1];
>>[x,u,y] = trim('trim_point', x0, u0);
```

则在MATLAB命令窗口中显示如下结果：

```
x =                % 系统所处平衡点处的状态变量值
1.0e-012 *
0.1716
```

0.0840

u =　　　　　　　　　　% 系统在平衡点处的输入值

0.3333

0.0000

y =　　　　　　　　　　% 系统在平衡点处的输出值

-1.0000

0.3333

8.5.3 非线性系统的线性化处理

迄今为止，线性系统的设计与分析技术已经非常完善了。但在实际的系统中，很少有真正的线性系统，大部分的系统都是非线性系统，然而对非线性系统的设计与分析还处于发展的初期，其设计与分析还主要依赖于设计者的设计经验。

1. 使用语法

```
[A,B,C,D] = linmod('sys', x, u)

[num,den] = linmod('sys', x, u)

sys_struc = linmod('sys', x, u)

[Ad,Bd,Cd,Dd] = dlinmod('sys', Ts, x, u)

[numd,dend] = dlinmod('sys', Ts, x, u)
```

2. 功能描述

[A,B,C,D] = linmod('sys', x, u)：在指定的系统状态x与系统输入u下对系统sys进行线性化处理，x与u的缺省值为0。A、B、C与D为线性化后的系统状态空间描述矩阵。

[num,den] = linmod('sys', x, u)：num与den为线性化后的系统传递函数描述。

sys_struc = linmod('sys', x, u)：返回线性化后的系统结构体描述，其中包括系统状态名称、输入与输出名称以及操作点的信息。

[Ad,Bd,Cd,Dd] = dlinmod('sys', Ts, x, u)：可以对非线性、多速率混合系统（包括离散系统与连续系统）进行线性化处理。其中Ts为系统的采样时间，Ts=0表示将离散系统线性化为连续系统。返回线性化后系统的状态控制描述。

[numd,dend] = dlinmod('sys', Ts, x, u)：返回线性化后系统的传递函数描述。

3. 应用举例

这里仍以蹦极跳系统为例说明非线性系统的线性化处理，从蹦极跳系统的系统动力学方程可知，蹦极跳系统为一非线性系统。下面对其进行线性化处理（其中系统模型中参数符合安全要求，即弹性常数为27），操作如下所示：

```
>> [A,B,C,D]=linmod('bungee_jumping_command')
% 返回线性化后系统的状态空间描述
```

```
A =

    0    1.0000

    0   -0.0143

B =

   Empty matrix: 2-by-0

C =

   -1.0000         0

D =

   Empty matrix: 1-by-0
```

% 返回线性化后系统的结构体描述

```
sys_struction =
            a: [2x2 double]
            b: [2x0 double]
            c: [-1.0000 0]
            d: [1x0 double]
    StateName: {2x1 cell}
   OutputName: {'bungee_jumping/Out1'}
    InputName: {0x1 cell}
    OperPoint: [1x1 struct]
           Ts: 0
```

此结构体内容包括了线性化后系统的如下信息：系统的状态空间描述矩阵a、b、c及d，系统模型中的状态变量名称StateName，系统输出结果名称OutputName，系统输入信号名称InputName，系统的操作点OperPoint以及系统采样时间Ts等。用户可以从此结构体变量中获得需要的信息，如

```
>> statename=sys_struction.StateName
    %获得系统模型中状态变量名称
statename =
        'bungee_jumping/position'
        'bungee_jumping/velocity'
```

8.6 回 调 函 数

1. 回调函数的基本概念

所谓回调函数，一般是指系统模型或系统模型中的某些模块在特定的时刻所运行的一系列用户自定义的命令的集合。对于系统模型与其中的系统模块而言，它们拥有不同的回调函数。这里特定的时刻一般是指系统中发生特定事件的时刻，如打开系统模型、对系统模型进行仿真、对模型中的模块进行操作以及打开模块等等。

2. 回调函数的设置与使用举例

使用MATLAB脚本命令对蹦极跳系统进行仿真分析，要求使用回调函数设置系统模型中的参数，并在仿真结束后绘制系统的相平面图（系统中两个状态之间的关系图，即蹦极者速度相对于其位置的关系图）。为了绘制系统的相平面图，需要输出系统中的状态变量，因此可以使用Simulink的参数设置对话框中的Workspace I/O选项设置输出状态（选中States复选框即可）。然后依次在MATLAB命令窗口中键入如下的命令：

```
>>set_param('bungee_jumping_command','InitFcn','m=70;g=10;a1=1;a2=1;k=27;');
```

% 'InitFcn'表示在系统仿真开始的时候开始调用，以设置模型参数

```
>>set_param('bungee_jumping_command','StopFcn',
'plot(xout(:,1),xout(:,2))');
```

% 'StopFcn'表示在系统仿真结束时开始调用，以绘制相平面图

```
>>sim('bungee_jumping_command',[0 100]);
```

% 完成系统仿真

3. 其它回调函数简介

1) 模型回调函数

CloseFcn：在系统模型框图关闭之前执行。

PostLoadFcn：在系统模型加载完成后执行。当编写一个要求模型完全加载后方能启动的界面程序时非常有用。

InitFcn：在系统模型仿真开始的时候调用。使用此回调函数可以在系统仿真开始时对系统模型中的参数进行设置。

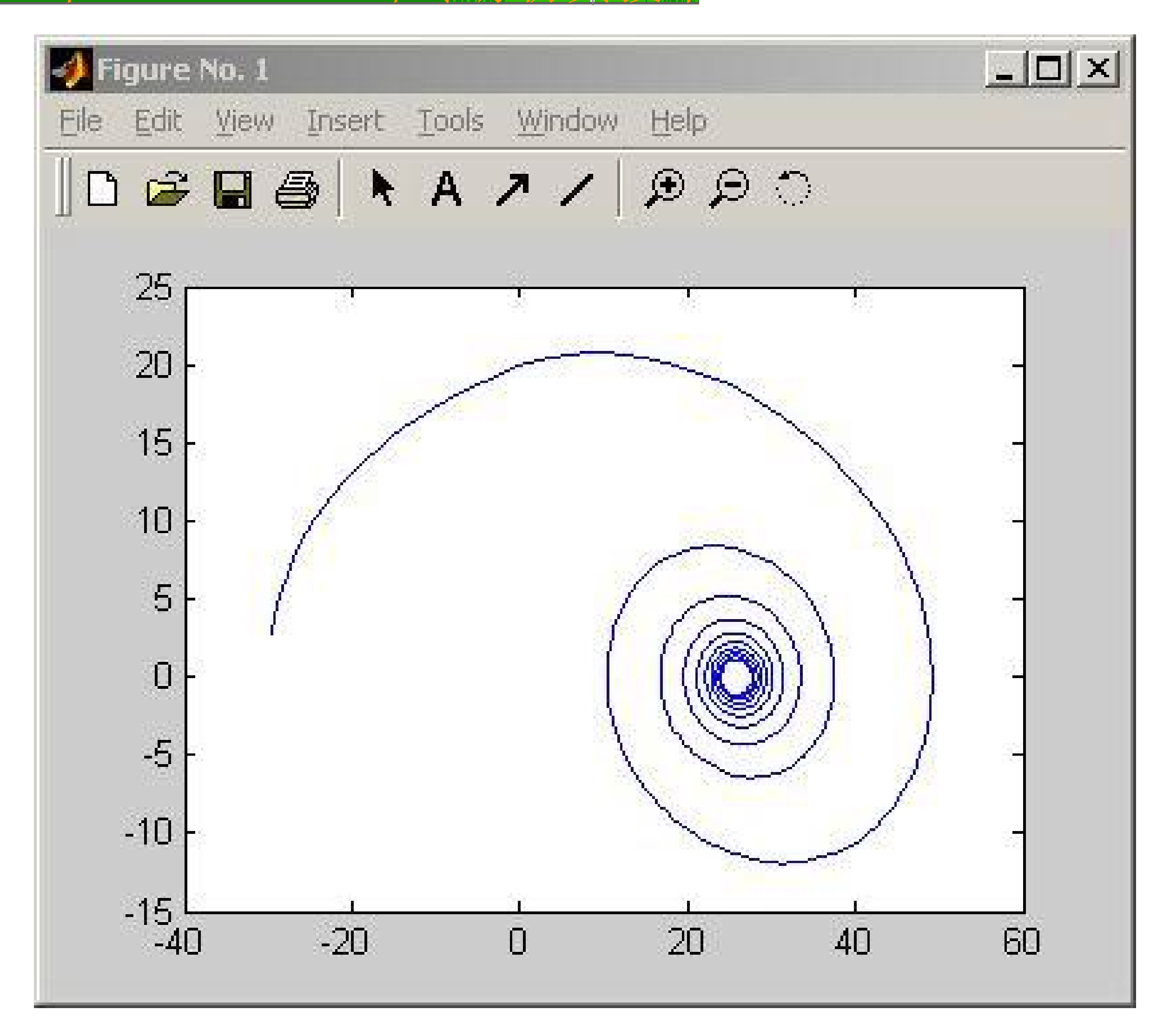


图8.17 蹦极跳系统的相平面图

PostSaveFcn：在系统模型保存后执行。

PreLoadFcn：在系统模型加载之前执行。

PreSaveFcn：在系统模型保存之前执行。

StartFcn：在系统仿真开始之前执行。

StopFcn：在系统仿真结束后执行。注意，在StopFcn执行之前系统仿真结果已经输出到MATLAB工作空间或是数据文件中。

2) 系统模块回调函数

CloseFcn：在使用close_system命令关闭系统模块时执行。

CopyFcn：在一个系统模块被复制后执行。此函数对于子系统中的模块是递归调用的。

DeleteFcn：在系统模块被删除之前执行。此函数对于子系统中的模块也是递归调用的。

DestroyFcn：在系统模块被清除后执行。

InitFcn：在系统框图被编译以及系统模块参数被求值之前执行。

LoadFcn：在系统框图加载后执行。此函数对于子系统中的模块是递归调用的。

ModelCloseFcn：在系统框图关闭前执行。此函数对于子系统中的模块是递归调用的。

MoveFcn：在系统模块移动或改变尺寸时执行。

NameChangeFcn：在系统模块的名称（或是路径）改变时执行。此函数对于子系统模块中的模块是递归调用的。

OpenFcn：在系统模块打开时运行。此函数通常与子系统模块结合使用。例如，在双击一个子系统模块或将其作为参数来调用open_system命令时运行。OpenFcn回调函数可以改变通常模块打开时的行为方式（即打开模块参数设置对话框）。

ParentCloseFcn：在关闭包含此系统模块的子系统之前（或作为通过new_system命令建立的新的子系统中的一部分时）执行。

PreSaveFcn：在系统框图保存前执行。对子系统中的模块递归调用执行。

PostSaveFcn：在系统框图保存后执行。对子系统中的模块递归调用执行。

StartFcn：在系统框图被编译之后，系统仿真开始之前执行。对于S-函数模块，StartFcn 恰好在第一次执行模块的mdlProcessParameterS-函数前执行。

StopFcn：在任何系统仿真终止的时候执行。对于S-函数块，StopFcn 在模块的 mdlTerminate函数执行之后执行。

UndoDeleteFcn：在被删除的系统模块被恢复后执行。

合理地使用Simulink系统模型与系统模块的回调函数，可以完成许多单独使用命令行所不易完成的任务，从而进一步提高系统设计与仿真的效率。希望用户能够在使用的过程中仔细体会命令行仿真方式的使用，进而熟练掌握命令行仿真技术以更好地对系统进行设计与仿真分析。

第9章　　S-函　数



9.1　S-函数概述

9.1.1 S-函数的基本概念

S-函数是系统函数(System Function) 的简称，是指采用非图形化的方式（即计算机语言，区别于Simulink的系统模块）描述的一个功能块。用户可以采用MATLAB代码，C，C++，FORTRAM或Ada等语言编写S-函数。S-函数由一种特定的语法构成，用来描述并实现连续系统、离散系统以及复合系统等动态系统；S-函数能够接收来自Simulink求解器的相关信息，并对求解器发出的命令做出适当的响应，这种交互作用非常类似于Simulink系统模块与求解器的交互作用。

S-函数作为与其他语言相结合的接口，可以使用这个语言所提供的强大能力。例如，MATLAB语言编写的S-函数可以充分利用MATLAB所提供的丰富资源，方便地调用各种工具箱函数和图形函数；使用C语言编写的S-函数则可以实现对操作系统的访问，如实现与其它进程的通信和同步等。

简单来说，用户可以从如下的几个角度来理解S-函数：

(1) S-函数为Simulink的“系统”函数。

(2) 能够响应Simulink求解器命令的函数。

(3) 采用非图形化的方法实现一个动态系统。

(4) 可以开发新的Simulink模块。

(5) 可以与已有的代码相结合进行仿真。

(6) 采用文本方式输入复杂的系统方程。

(7) 扩展Simulink功能。M文件S-函数可以扩展图形能力，C MEX S-函数可以提供与操作系统的接口。

(8) S-函数的语法结构是为实现一个动态系统而设计的（默认用法），其它S-函数的用法是默认用法的特例（如用于显示目的）。

9.1.2 如何使用S-函数

前面简单介绍了S-函数的基本概念及其主要的功能。在动态系统设计、仿真与分析中，用户可以使用Functions & Tables模块库中的S-function模块来使用S-函数；S-function模块是一个单输入单输出的系统模块，如果有多个输入与多个输出信号，可以使用Mux模块与Demux模块对信号进行组合和分离操作。

一般而言，S-函数的使用步骤如下：

(1) 创建S-函数源文件。创建S-函数源文件有多种方法，当然用户可以按照S-函数的语法格式自行书写每一行代码，但是这样做容易出错且麻烦。Simulink为我们提供了很多S-函数模板和例子，用户可以根据自己的需要修改相应的模板或例子即可。

(2) 在动态系统的Simulink模型框图中添加S-function模块，并进行正确的设置。

(3) 在Simulink模型框图中按照定义好的功能连接输入输出端口。


S-函数对话框
S-函数源文件
S-function块
my_sfunction
S-Function
Block Parameters: S-Function
S-Function
User-definable block. Blocks may be written in M, C, Fortran or Ada and must conform to S-function standards. t,x,u and flag are automatically passed to the S-function by Simulink. "Extra" parameters may be specified in the 'S-function parameters' field.
Parameters
S-function name:
my_sfunction
S-function parameters:
param1,param2,param3
OK
Cancel
Help
Apply
/*
* my_sfunction
*
*/
#define S_FUNCTION_NAME my_sfunction
#define S_FUNCTION_LEVEL 2
#include "simstruc.h"
/*====================*
* S-function methods *
====================/
/* Function: mdlInitializeSizes */
static void mdlInitializeSizes(SimStruct *S)
{
。
。
。

图9.1　S-函数块对话框

【例9.1】 使用S-函数实现系统：y=2u。

解：(1) 打开模板M文件S-函数模板文件sfuntmpl.m，在\MATLABroot\work目录下另存为doublesfunction.m。

(2) 找到函数mdlInitializeSizes，修改以下代码：

```
sizes.NumOutputs = 1;
sizes.NumInputs = 1;
```

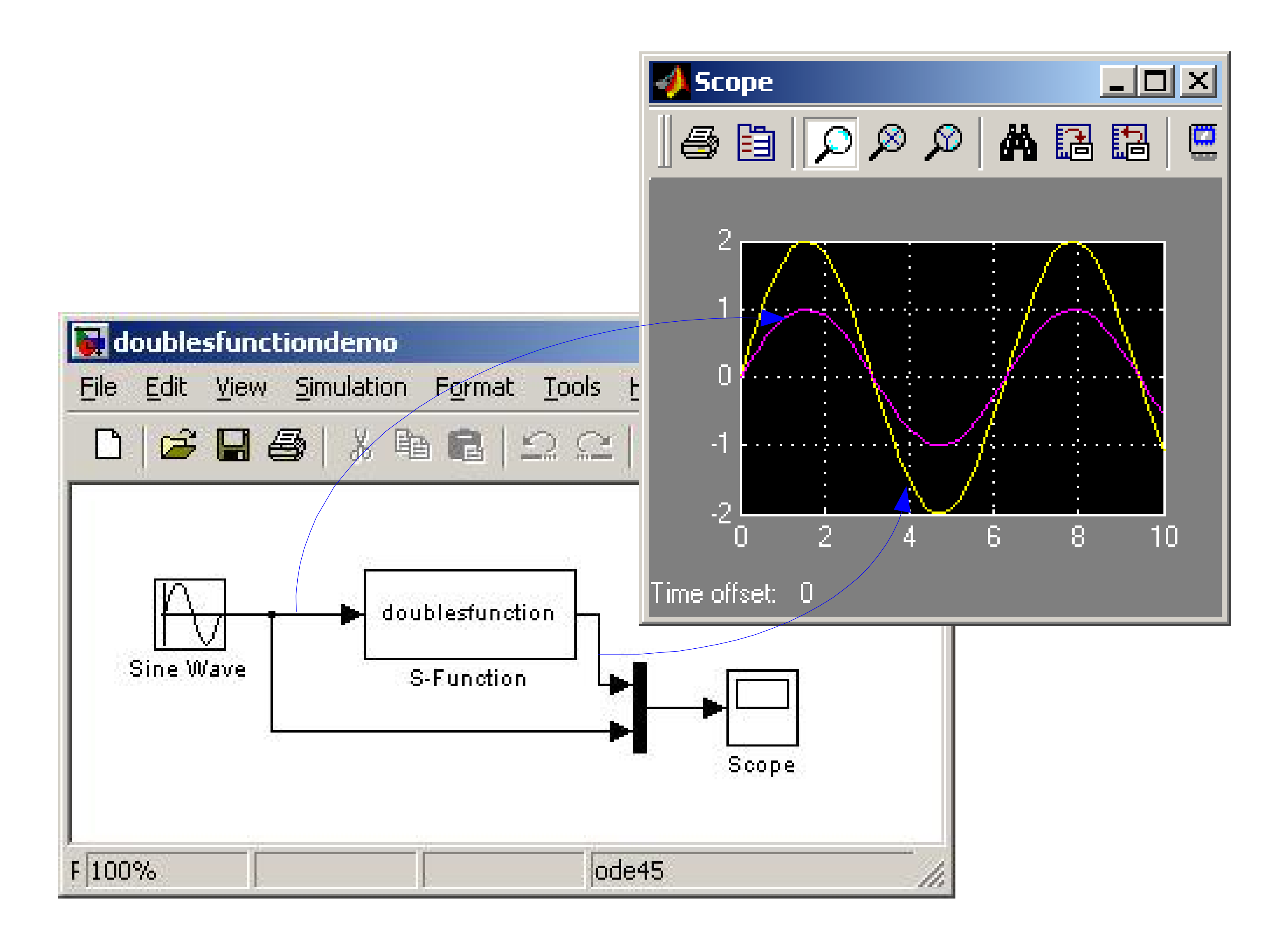


图9.2　一个用S-函数实现的简单系统

(3) 找到函数mdlOutputs，加入以下代码：

```
sys=2*u;
```

(到现在为止我们的第一个S-函数写完了。下面演示一下它的作用。)

(4) 在Simulink空白页中添加S-function 块，打开S-function 块对话框，参数S-function name设置为doublesfunction 。按照图9.2添加连接好其余的各个模块。

(5) 开始仿真，在Scope中观察输出结果，可以看到输入正弦信号被放大为原来的2倍，如图9.2所示。

9.1.3 与S-函数相关的一些术语

理解下列与S-函数相关的一些基本术语对于用户理解S-函数的概念与编写都是非常有益的；而且这些概念在其它的仿真语言中也是会经常遇到的。

1. 仿真例程（Routines）

Simulink在仿真的特定阶段调用对应的S-函数功能模块（函数），来完成不同的任务，如初始化、计算输出、更新离散状态、计算导数、结束仿真等，这些功能模块（函数）称为仿真例程或者回调函数（call back functions）。表9.1列出了S-函数例程函数和对应的仿真阶段。关于仿真例程将在S-函数工作原理一节详细介绍。

表9.1 S-函数例程

S-函数仿真例程	仿真阶段
mdlInitialization	初始化
mdlGetTimeofNextVarHit	计算下一个采样点
mdlOutput	计算输出
mdlUpdate	更新离散状态
mdlDerivatives	计算导数
mdlTerminate	结束仿真

2. 直接馈通（Direct feedthrough）

直接馈通意味着输出或可变采样时间与输入直接相关（详见第6章）。在如下的两种情况下需要直接馈通：

(1) 某一时刻的系统输出y中包含某一时刻的系统输入u。

(2) 系统是一个变采样时间系统（variable sample time system）且采样时间计算与输入u相关。

$$time = (n \times sample_time_value) + offset_time$$

其中*n*表示第*n*个采样点。

Simulink在每一个采样点上调用mdlOutput和mdlUpdate例程。对于连续时间系统采样时间和偏移量的值应该设置为零。采样时间还可以继承自驱动模块、目标模块或者系统最小采样时间，这种情况下采样时间值应该设置为-1，或者INHERITED_SAMPLE_TIME。

4. 动态输入（Dynamically sized inputs）

S-函数支持动态可变维数的输入。S-函数的输入变量u的维数决定于驱动S-函数模块的输入信号的维数。



9.2　S-函数的工作原理

9.2.1 状态方程

在对动态系统建模时，总是能够采用广义的状态空间形式对无论是线性系统还是非线性系统进行描述。这个描述包含以下两个方程：

状态方程：

输出方程：

状态方程描述了状态变量的一阶导数与状态变量、输入量之间的关系。*n*阶系统具有*n*个独立的状态变量，系统状态方程则是*n*个联立的一阶微分方程或者差分方程。对于一个系统，由于所选择的状态变量不同，会导出不同的状态方程，因此状态方程的形式不是唯一的。输出方程描述了输出与状态变量、输入量之间的关系。输出量根据任务的需要确定。一个典型的线性系统的状态方程可以用矩阵的形式描述为：

状态方程：

输出方程：

其中***A***、***B***、***C***、***D***分别是状态矩阵、输入矩阵、输出矩阵、前馈矩阵。

Simulink框图的大部分模块都具有一个输入向量***u***、一个输出向量***y***和一个状态向量***x***，如图9.3所示。

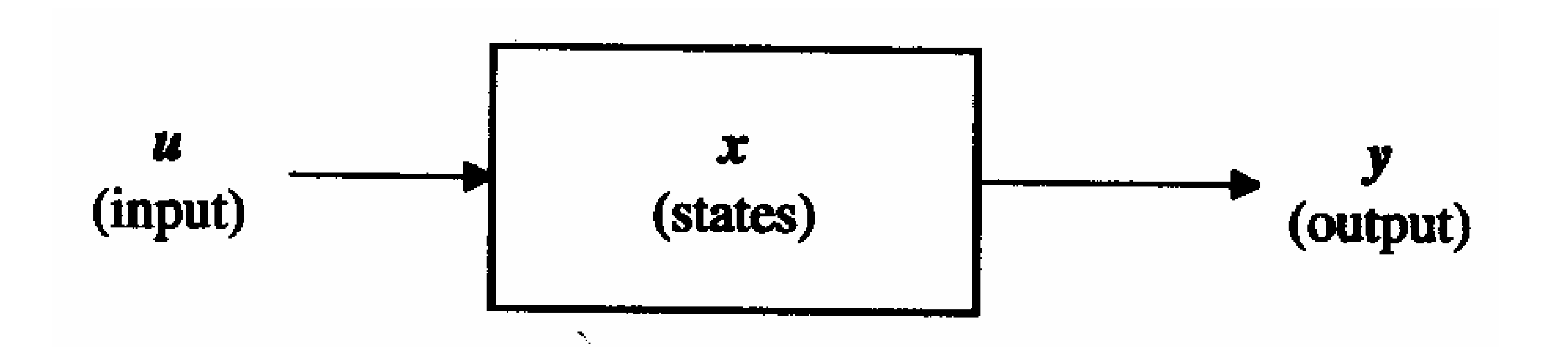


图9.3 Simulink模块

$\boldsymbol{u}$，$\boldsymbol{x}$，$\boldsymbol{y}$和时间$\boldsymbol{t}$之间存在如下关系：

输出方程：

连续状态方程：

离散状态方程：

其中。

S-函数同样是一个Simulink模块。它的以下几个例程函数清楚地体现了状态空间所描述的特性。

(1)　S-函数中的连续状态方程描述。状态向量的一阶导数是状态***x***、输入***u***和时间***t***的函数。在S-函数中，状态的一阶导数是在mdlDerivatives例程中计算的，并将结果返回供求解器积分。

(2)　S-函数中的离散状态方程描述。下一步状态的值依赖于当前的状态输入***u***和时间***t***。这是通过mdlUpdate例程完成的，并将结果返回供求解器在下一步时使用。

(3)　S-函数中的输出方程描述。输出值是状态、输入和时间的函数。

9.2.2 Simulink仿真的两个阶段

理解S-函数首先要很好地了解Simulink的仿真过程。仿真包含两个主要阶段，第一个阶段是初始化，这时块的所有参数都已确定下来。初始化阶段完成了以下工作：

(1) 传递参数给MATLAB进行求值。

(2) 得到的数值作为实际的参数使用。

(3) 展开模型的层次，每个子系统被它们所包含的块替代。

(4) 检查信号的宽度和连接。

(5) 确定状态初值和采样时间。

仿真运行阶段的工作可以概括为：

(1) 计算输出。

(2) 更新离散状态。

(3) 计算连续状态，连续状态的计算过程：

① 每个块按照预先确定的顺序计算输出。

② 每个块使用当前时间、块的输入和状态计算它的导数。

③ 导数返回给求解器，通过积分得到下一步状态的值。

(4) 计算输出，过零可能被激活。

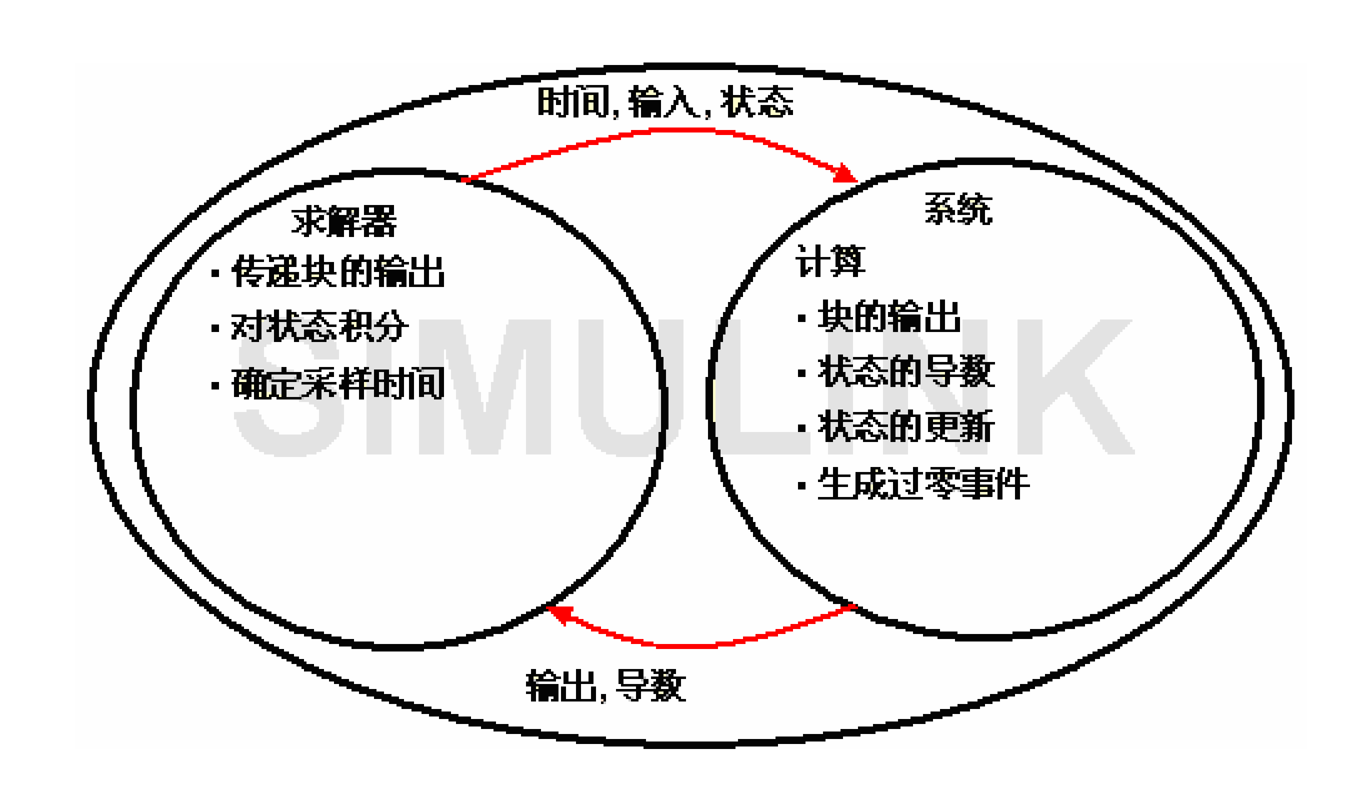


图9.4 求解器与系统的交互作用关系

9.2.3 S-函数仿真流程

S-函数是Simulink的重要组成部分，它的仿真过程包含在Simulink仿真过程之中，所以上一节所述同样适用于S-函数。如图9.5所示，S-函数的仿真流程也包括初始化阶段和运行阶段两个阶段。图9.5中每个功能模块都对应于一个仿真例程或者回调函数。

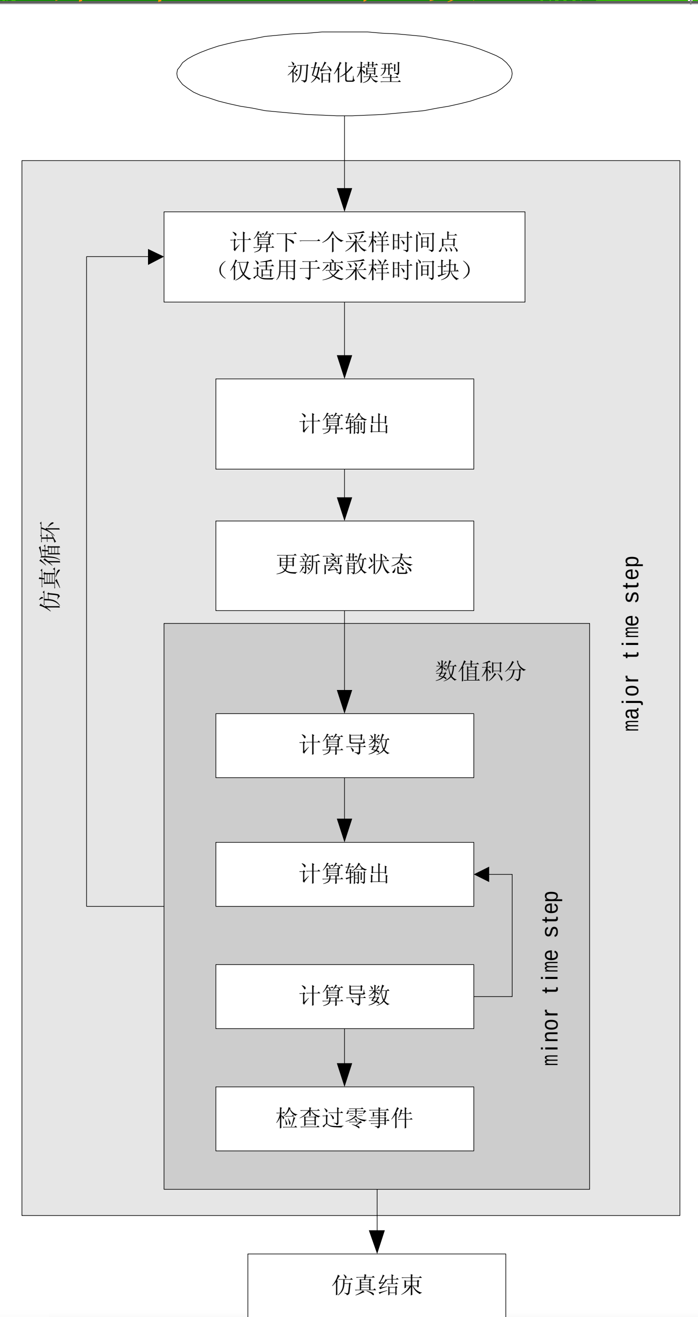

初始化模型
计算下一个采样时间点
（仅适用于变采样时间块）
计算输出
更新离散状态
仿真循环
major time step
数值积分
计算导数
计算输出
计算导数
minor time step
检查过零事件
仿真结束

图9.5 S-函数仿真流程

(1) 初始化：在仿真开始前，Simulink在这个阶段初始化S-函数。

① 初始化结构体SimStruct，它包含了S-函数的所有信息。

② 设置输入输出端口数。

③ 设置采样时间。

④ 分配存储空间。

(2) 计算下一个采样时间点：只有在使用变步长求解器进行仿真时，才需要计算下一个采样时间点，即计算下一步的仿真步长。

(3) 计算输出：计算所有输出端口的输出值。

(4) 更新状态：此例程在每个步长处都要执行一次，可以在这个例程中添加每一个仿真步都需要更新的内容，例如离散状态的更新。

(5) 数值积分：用于连续状态的求解和非采样过零点。如果S-函数存在连续状态，Simulink就在minor step time内调用mdlDdrivatives和mdlOutput两个S-函数例程。



9.3 编写M文件S-函数

9.3.1 M文件S-函数的工作流程

M文件S-函数和上节所介绍的S-函数仿真流程是一致的。它调用例程函数的顺序是通过标志Flag来控制的。图9.6给出了各仿真阶段的标志值、变量值及其对应仿真例程。

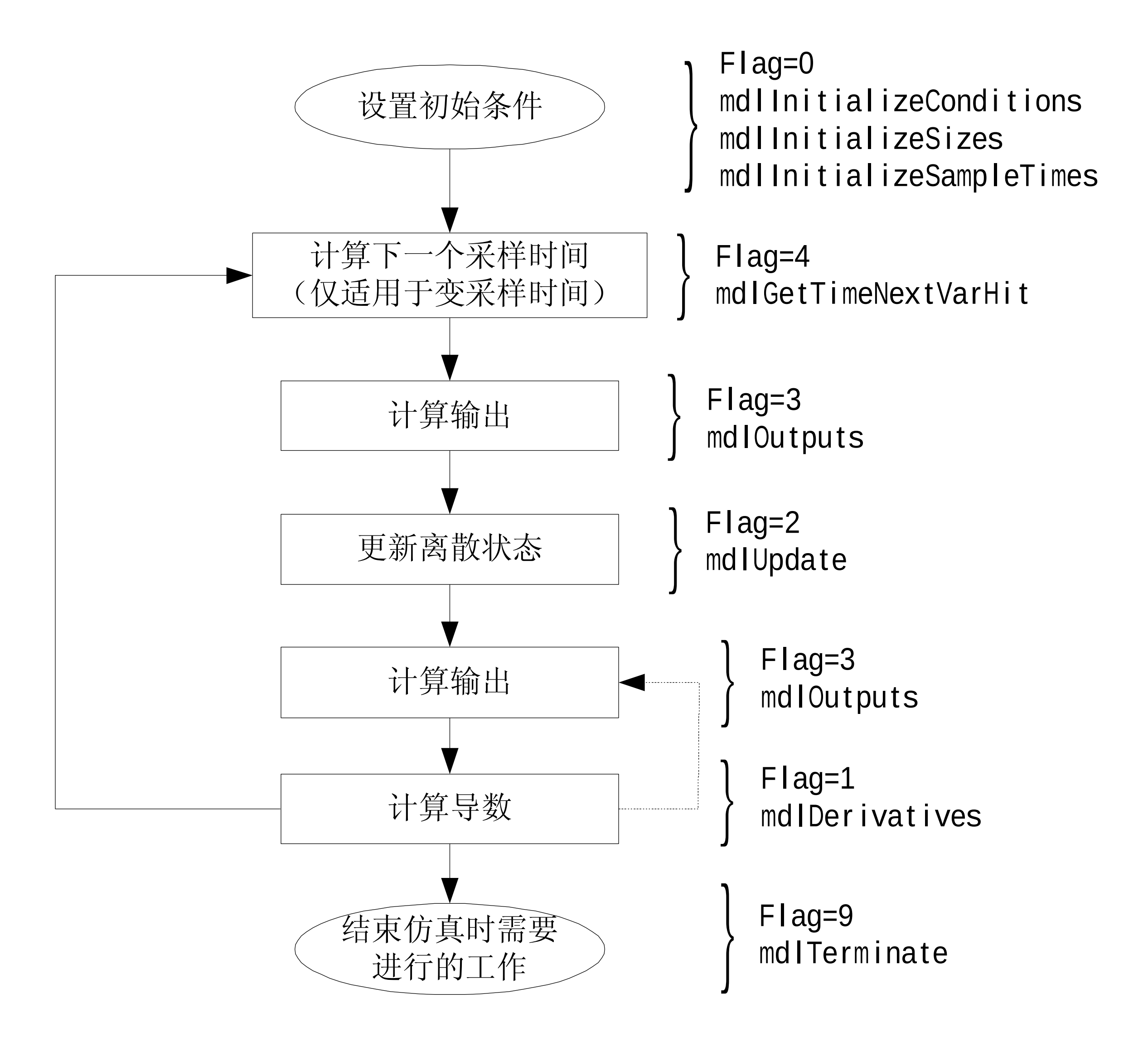


图9.6 M文件S-函数流程

9.3.2 M文件S-函数模板

Simulink为我们编写S-函数提供了各种模板文件，其中定义了S-函数完整的框架结构，用户可以根据自己的需要加以剪裁。编写M文件S-函数时，推荐使用S-函数模板文件sfuntmpl.m。这个文件包含了一个完整的M文件S-函数，它包含1个主函数和6个子函数。在主函数内程序根据标志变量Flag，由一个开关转移结构（Switch-Case）根据标志将执行流程转移到相应的子函数，即例程函数。Flag标志量作为主函数的参数由系统（Simulink引擎）调用时给出。了解这个模板文件的最好方式莫过于直接打开看看其代码。

要打开模板文件，可在MATLAB命令行下输入：

>>edit sfuntmpl

或 者 双 击 \S-function demos\M-file S-functions\M-file template 块。

下面是删除了其原有注释的代码，结构反而更为清晰紧凑。

% 主函数

% 主函数包含四个输出：sys 数组包含某个子函数返回的值，它的含义随着调用子函数的不同而

% 不同；x0 为所有状态的初始化向量；str是保留参数，总是一个空矩阵；Ts返回系统采样时间。

% 的仿真流程位置如图9.6所示；此外输入参数后面还可以接续一系列的附带参数。另外别忘了

% 编写自己的S-函数时，应该把函数名sfuntmpl改为S-function块中对应的函数名

```
function [sys,x0,str,ts] = sfuntmpl(t,x,u,flag)
switch flag
 case 0
  [sys,x0,str,ts]=mdlInitializeSizes;
 case 1
  sys=mdlDerivatives(t,x,u);
 case 2
  sys=mdlUpdate(t,x,u);
 case 3
```

```
 sys=mdlOutputs(t,x,u);
  case 4
    sys=mdlGetTimeOfNextVarHit(t,x,u);
  case 9
    sys=mdlTerminate(t,x,u);
  otherwise
    error(['Unhandled flag = ',num2str(flag)]);
end
% 主函数结束
% 下面是各个子函数，即各个仿真例程
% 1. 初始化例程子函数：提供状态、输入、输出、采样时间数目和
     初始状态的值。必须有该子函
```

% 数。初始化阶段，标志变量首先被置为 0，S-函数被第一次调用时，mdlInitializeSizes 子函数

% 首先被调用。这个子函数应当为系统提供S-function块的下列信息：

% 连续状态的个数

% 离散状态的个数

% 输出的个数

% 输入的个数

% 是否直接馈通：这是一个布尔量，当输出值直接依赖于同一时刻的输入值时为 1；否则为0

% 采样时间的个数：每个系统至少有一个采样时间

% 这些信息是通过一个数据结构sizes来表示的

% 在该函数中用户还应该提供初始状态 x0，采样时间 ts。ts 是一个 m×2 的矩阵，其中第 k 行

% 包含了对应与第 k 个采样时间的采样周期值和偏移量。另外，在该子函数中str 设置为空：[ ]，

% str是保留变量，暂时没有任何意义

```
function [sys,x0,str,ts]=mdlInitializeSizes
sizes = simsizes;                 % 生成sizes数据结构
sizes.NumContStates  = 0;         % 连续状态数，缺省为 0
sizes.NumDiscStates  = 0;         % 离散状态数，缺省为 0
sizes.NumOutputs     = 0;         % 输出量个数，缺省为 0
```

```
sizes.NumInputs      = 0;              % 输入量个数，缺省为 0
sizes.DirFeedthrough = 1;              % 是否存在直接馈通。1：存在；
    0：不存在，缺省为 1
sizes.NumSampleTimes = 1;                          % 采样时间个数，至少是
    一个
sys = simsizes(sizes);        % 返回sizes数据结构所包含的信息

x0  = [  ];                                              % 设置初始状态
str = [  ];                            % 保留变量置空
ts  = [0 0];          % 采样时间：[采样周期 偏移量]，采样周期为0表示
    是连续系统
```

% 2. 计算导数例程子函数：给定 t,x,u, 计算连续状态的导数，用户应该在此给出系统的连续状态

% 方程。该子函数可以不存在

```
function sys=mdlDerivatives(t,x,u)
sys = [  ];     % sys 表示状态导数，即dx
```

% 3. 状态更新例程子函数：给定 t,x,u, 计算离散状态的更新。每个仿真步长必然调用该子函数，

% 不论是否有意义。用户除了在此描述系统的离散状态方程外，还可以填入其它每个仿真步长都

% 有必要执行的代码

```
function sys=mdlUpdate(t,x,u)
```

sys = [ ]; % sys 表示下一个离散状态 即 x(k+1)

% 4. 计算输出例程子函数：给定 t,x,u, 计算输出。该子函数必须存在，用户可以在此描述系统的

% 输出方程

function sys=mdlOutputs(t,x,u)

sys = [ ]; % sys 表示下输出，即 y

% 5. 计算下一个采样时间，仅在系统是变采样时间系统时调用

function sys=mdlGetTimeOfNextVarHit(t,x,u)

```
sampleTime = 1;        % 设置下一次的采样时间是1  s以后
sys = t + sampleTime;    %   sys 表示下一个采样时间点


% 6. 仿真结束时要调用的例程函数，在仿真结束时调
   用，用户可以在此完成结束仿真所需的必要
% 工作
function sys=mdlTerminate(t,x,u)
sys = [  ];
```

9.3.3 含用户参数的简单系统

M文件S-函数除了几个必需的参数，还可以加入用户自定义参数，这些参数需要在S-函数的输入参数中列出。首先主函数要做适当的修改，以便将用户参数传递到子函数中，子函数的定义也应当进行相应的修改，以便通过输入参数接收用户的参数。在编写S-函数时，应当区分哪些参数会影响一个子函数的执行，然后针对这些参数做相应的改动。另外，使用这些参数时不要忘了在S-functions模块的模块参数对话框内输入它们。

【例9.2】 用S-函数实现gain模块。

解：增益值作为S-函数用户自定义参数由用户输入。

(1) 对M文件S-函数的主函数定义做了修改，增加新的参数，并采用新的函数名：

```
function [sys,x0,str,ts]=sfun_vargain(t,x,u,flag,gain)
```

(2) 由于增益参数只是用来计算输出值，因而对mdlOutputs的调用可修改成：

```
case 3
 sys=mdlOutputs(t,x,u,gain);
```

(3) 修改初始化例程：

```
sizes.NumContStates  = 1;
sizes.NumDiscStates  = 0;
sizes.NumOutputs     = 1;
sizes.NumInputs      = 0;
sizes.DirFeedthrough = 1;
x0＝0；
```

(4) mdlOutputs 子函数的定义也做了相应的修改，将增益作为参数输入：

```
function sys=mdlOutputs(t,x,u,g)
 sys = g*u;
```

输出通过增益和输入的乘积得到，并通过sys返回。从图9.7中可以清晰地看出用户自定义参数gain的来龙去脉。图中有两个S-函数块，一个是原始未封装的S-函数模块，一种是封装后的S-函数模块。

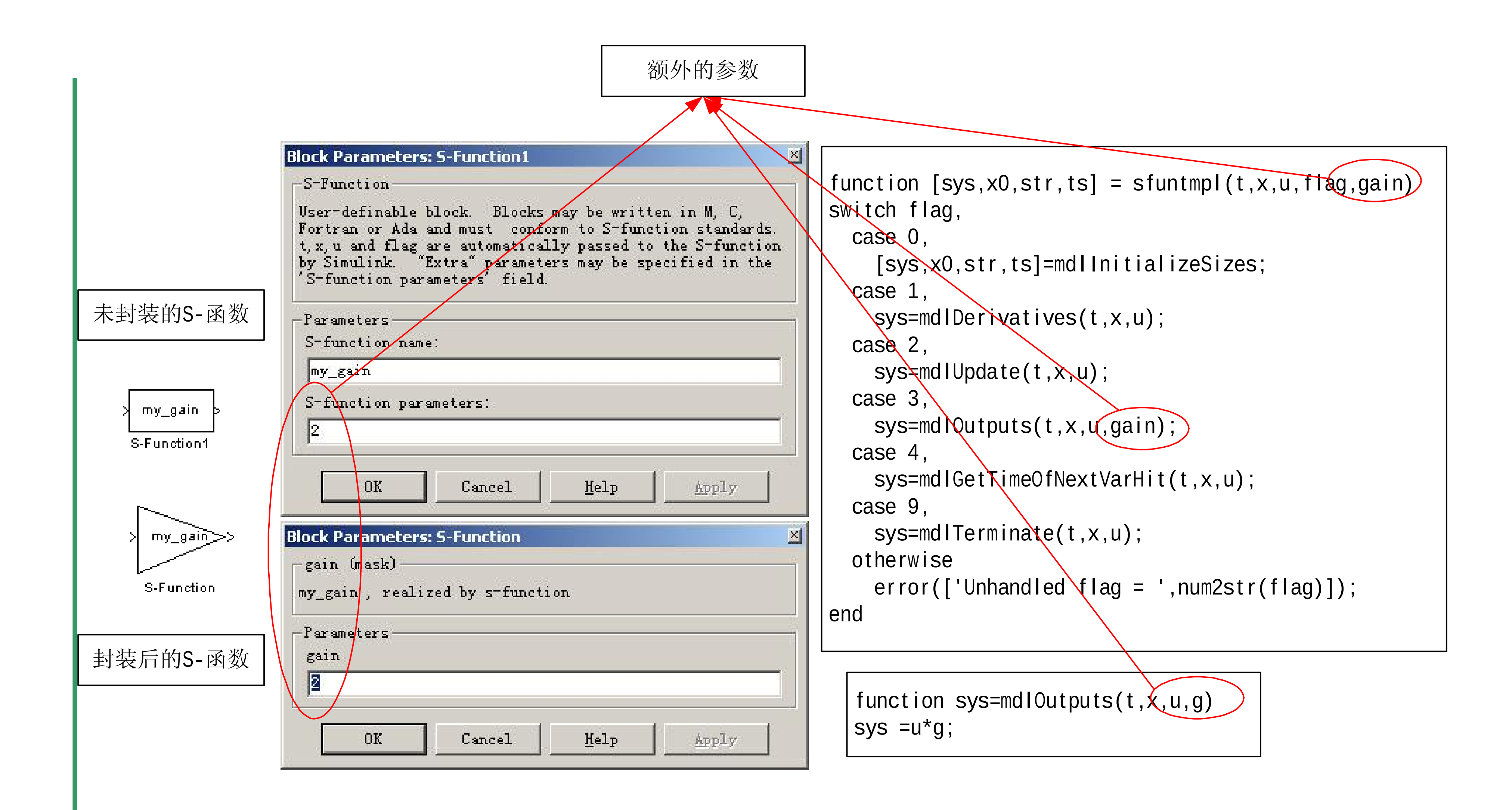


图9.7 含用户参数的简单系统

9.3.4离散系统的S-函数描述

用S-函数模板实现一个离散系统时，首先对mdlInitializeSizes子函数进行修改，声明离散状态的个数，对状态进行初始化，确定采样时间等。然后再对mdlUpdate和mdlOutputs函数做适当的修改，分别输入要表示的系统的离散状态方程和输出方程即可。

离散系统最简单的例子是单位延迟。单位延迟的差分方程为$y(k+1)=u(k)$，这相当于在每个仿真步长对采样做一阶保持。等价的状态方程表达为$x(k+1)=u(k)$，$y(k)=x(k)$。于是，mdlUpdate 和 mdlOutputs 应当修改为：

```
function sys=mdlUpdate(t,x,u)
sys = u;
function sys=mdlOutputs(t,x,u)
sys = x;
```

要观察该S-函数源文件，可在MATLAB命令行输入：

>> edit sfundsc2

然后双击\S-function demos\M-file S-functions\Unit delay观察仿真结果。图9.8是其Simulink框图。示波器显示正弦信号被采样并作一阶保持。用户不妨将Sine Wave块Sample time参数设置为其它值，再观察一下会有什么结果。

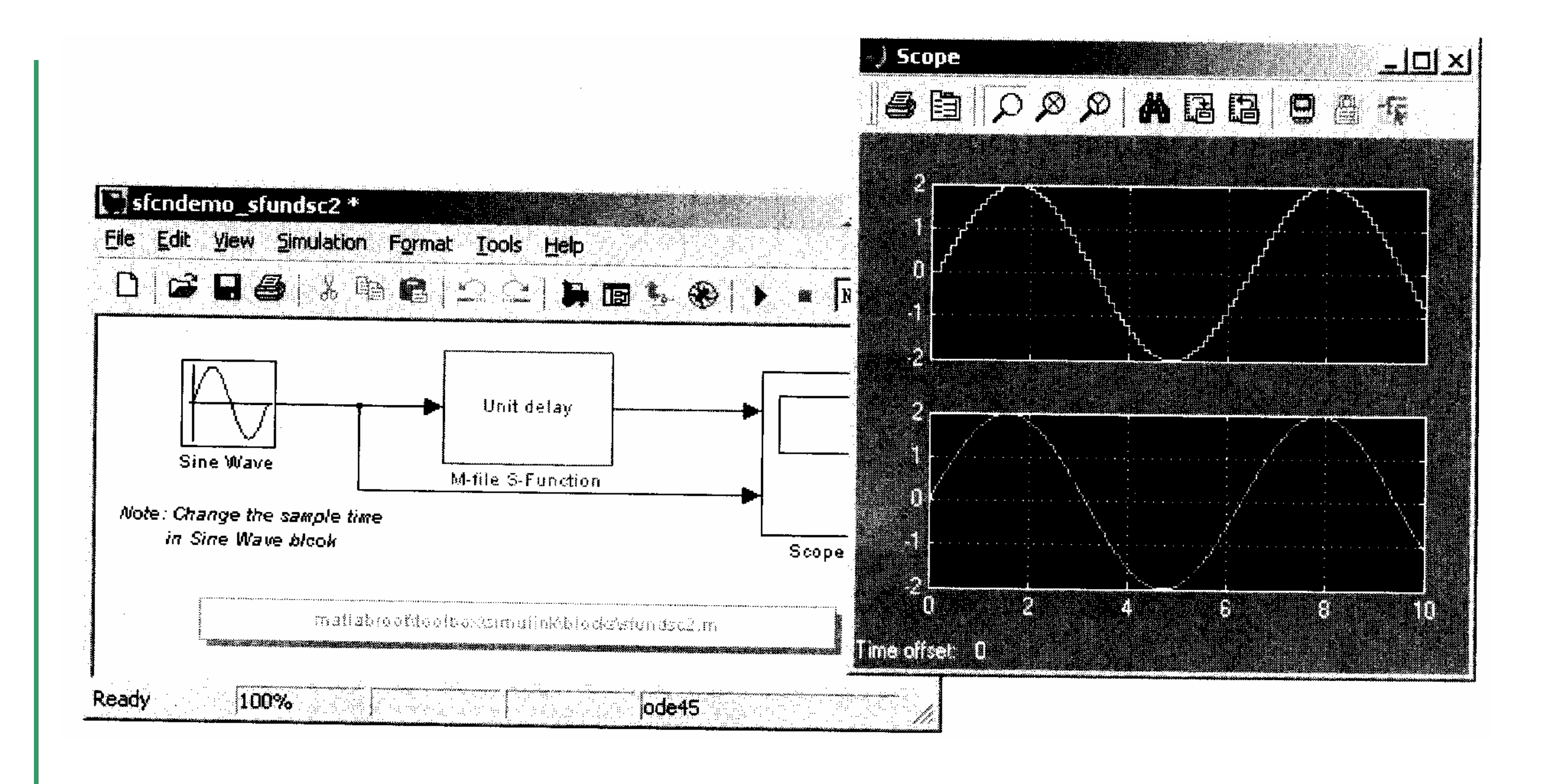


图9.8 用S-函数实现单位延迟

【例9.3】 编写一个 S-函数，说明人口的动态变化。设人口出生率为 r，资源为 K，初始人口数量为 init，则人口变化规律为：p(n)=r*p(n-1)*(1-p(n-1)/K)，p(0)=init。

解：(1) 修改M文件S-函数主函数：

```
function [sys,x0,str,ts]=sfun_population(t,x,u,flag,r,K,init)
case 0,
    [sys,x0,str,ts]=mdlInitializeSizes(init);
    …
    case 2,
    sys=mdlUpdate(t,x,u,r,K);
```

(2) 修改初始化部分:

```
function [sys,x0,str,ts]=mdlInitializeSizes(init)
sizes = simsizes;
sizes.NumContStates  =  0;
sizes.NumDiscStates  =  1;          % 一个离散状态, 人口数量
sizes.NumOutputs  =  1;    % 一个输出
sizes.NumInputs  =  0;
sizes.DirFeedthrough  =  0;         % 不存在直接馈通
sizes.NumSampleTimes  =  1;
sys = simsizes(sizes);
x0  = init;  % 初始状态：人口基数
…
```

(3) 在这个例子中，p为状态，输出等于状态，于是

```
function sys=mdlUpdate(t,x,u,r,K)
sys = [r*x*(1-x/K)];
function sys=mdlOutputs(t,x,u)
sys = [x];
```

设置初始人口基数init＝1e5,资源K＝1e6，人口出生率r＝1.05。仿真结果和Simulink框图如图9.9所示。可以看到在以上条件下人口数量随时间缓慢减少，直到稳定到一个值。当增加资源的量时，例如令K＝3e6，会看到人口数随时间的增加而增加，直到稳定到一个值，说明在此资源数下已经不能够再承载更多的人口。

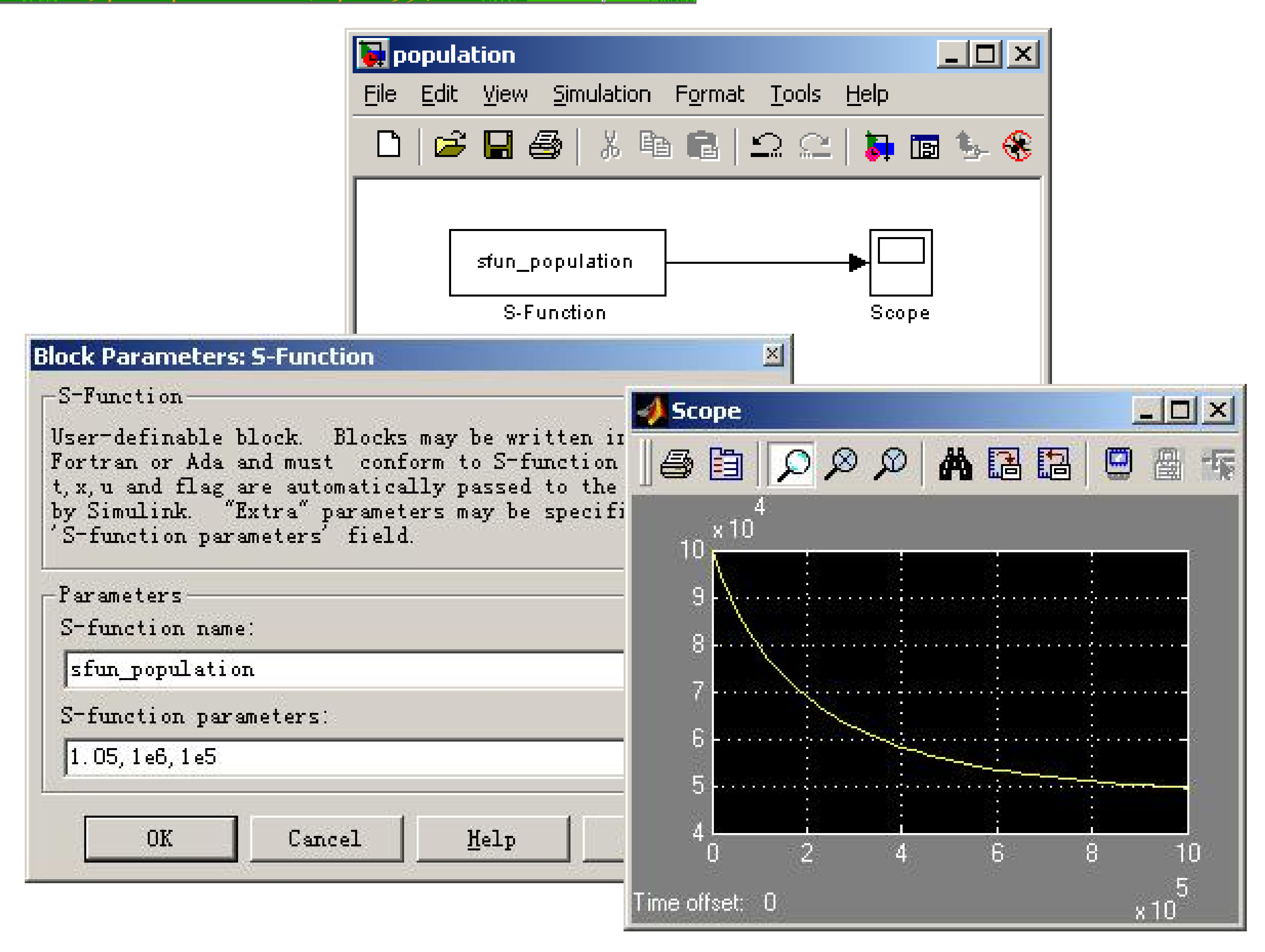


图9.9 编写S-函数仿真人口系统

9.3.5 连续系统的S-函数描述

用S-函数实现一个连续系统时，首先mdlInitilizeSizes子函数应当做适当的修改，包括确定连续状态的个数、状态初始值和采样时间设置。另外，还需要编写mdlDerivatives子函数，将状态的导数向量通过sys变量返回。

如果系统状态不止一个，可以通过索引x(1)，x(2)得到各个状态。当然，对于多个状态，就会有多个导数与之对应。在这种情况下，sys为一个向量，其中包含了所有连续状态的导数。与前文所述一样，修改后的mdlOutputs中应包含系统的输出方程。下面使用S-函数实现一个最简单的连续系统模块——积分器。

【例9.4】 用M文件S-函数实现一个积分器。

解：对于一个积分器，输入输出之间的关系：，令状态，则系统状态方程为，系统输出方程为。下面修改S-函数模板文件。

(1) 修改 S-函数模板的第一行：

```
function [sys,x0,str,ts] = sfun_int(t,x,u,flag,initial_state)
```

(2) 初始状态应当传递给mdlInitializeSizes：

```
case 0,
[sys,x0,str,ts]=mdlInitializeSizes(initial_state);
```

(3) 设置初始化参数：

```
function [sys,x0,str,ts]=mdlInitializeSizes(initial_state)
```

```
sizes = simsizes;
sizes.NumContStates  = 1;
sizes.NumDiscStates  = 0;
sizes.NumOutputs     = 1;
sizes.NumInputs      = 1;
sizes.DirFeedthrough = 0;
sizes.NumSampleTimes = 1;   % 采样时间个数，至少是一
    个
sys = simsizes(sizes);
x0  = initial_state;         % 初始化状态变量
```

(4) 书写状态方程：

```
function sys=mdlDerivatives(t,x,u)
sys = u;
```

(5) 添加输出方程：

```
function sys=mdlOutputs(t,x,u)
sys = x;
```

令输入，初始状态，则得到如图9.10所示的仿真结果。

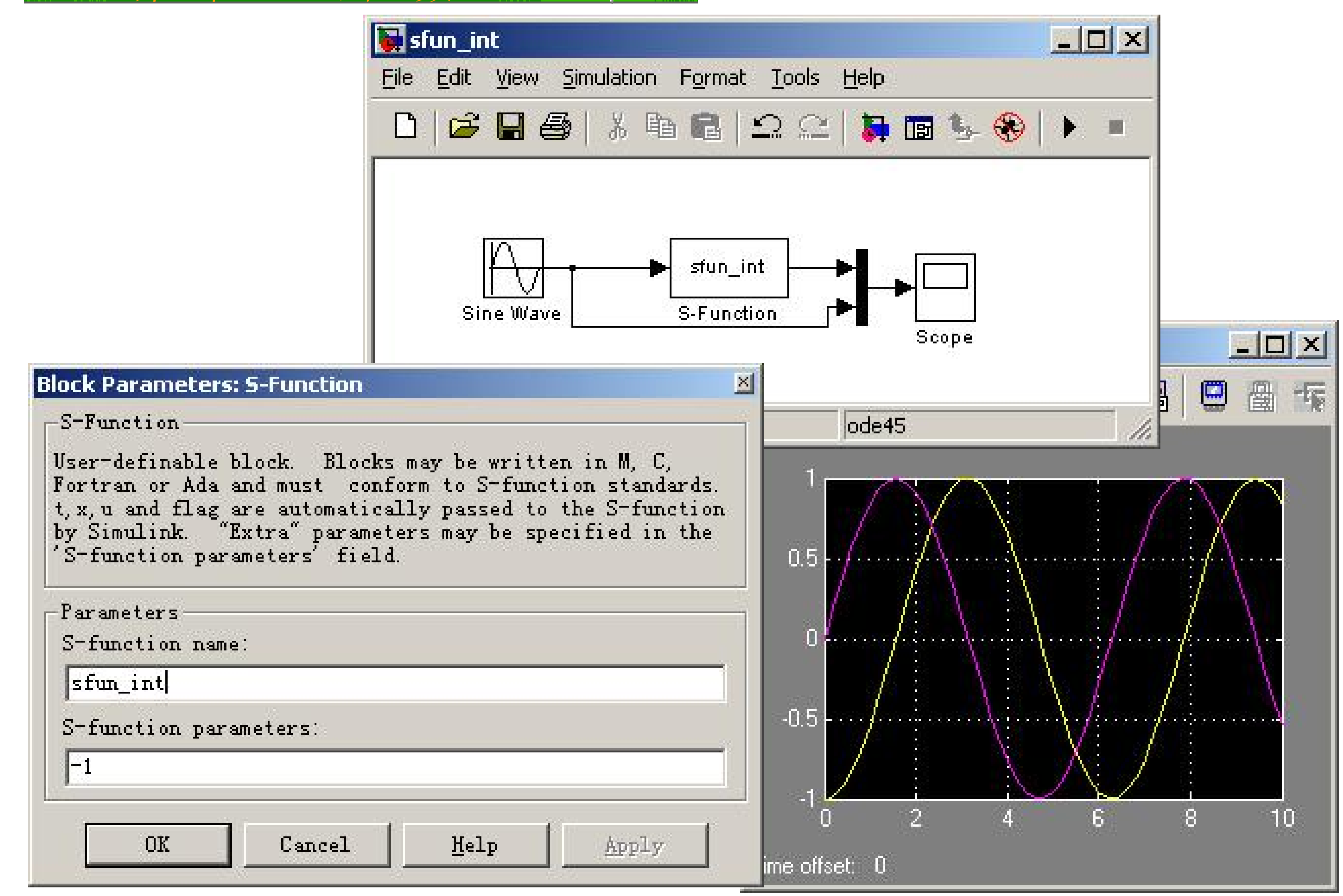


图9.10　用S-函数实现的积分器

9.3.6 混合系统的S-函数描述

所谓混合系统，就是既包含离散状态，又包含连续状态的系统。这里使用一个Simulink自带的例子mixedm.m来说明这个问题。mixedm.m描述了一个连续积分系统外加一个离散单位延迟。其Simulink框图如图9.11所示。在仿真的每个采样时间点上Simulink都要调用mdlUpdate、mdlOutput和mdlGetTimeOfNextVarHit仿真例程（如果是固定步长就不需要mdlGetTimeOfNextVarHit例程）；所以在mdlUpdate、mdlOutput中需要判断是否需要更新离散状态和输出。因为对于离散状态并不是在所有的采样点上都需要更新，否则就是一个连续系统了。

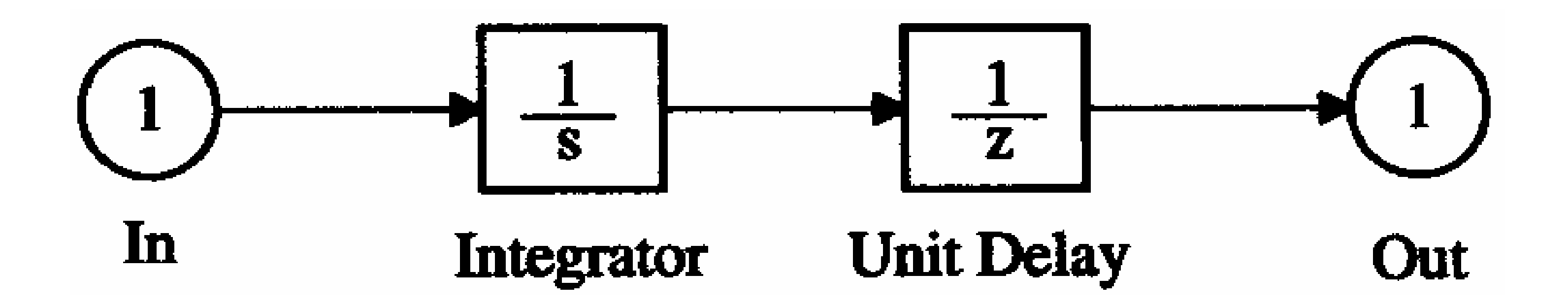


图9.11　一个简单的混合系统

在命令行输入：

```
>> edit mixedm.m
```

下面列出了部分代码：

```
function [sys,x0,str,ts] = mixedm(t,x,u,flag)
dperiod = 1;
doffset = 0;            % 设置离散采样周期和偏移量
. . .                   % switch-case 结构
function [sys,x0,str,ts]=mdlInitializeSizes(dperiod,doffset)
sizes = simsizes;
sizes.NumContStates  = 1;     % 一个连续状态
```

```
sizes.NumDiscStates  = 1;      % 一个离散状态
sizes.NumOutputs     = 1;      % 一个输出
sizes.NumInputs      = 1;      % 一个输入
sizes.DirFeedthrough = 0;      % 没有前馈
sizes.NumSampleTimes = 2;              % 两个采样时间
sys = simsizes(sizes);
x0  = ones(2,1);
str = [  ];
ts  = [0 0; dperiod doffset];   % 一个采样时间是 [0  0] 表示
    是连续系统
```

% 离散系统的采样时间就是在主程序开始所设置的两个变量

```
function sys=mdlDerivatives(t,x,u)
sys = u;                    % 连续系统是一积分环节

function sys=mdlUpdate(t,x,u,dperiod,doffset)
if abs(round((t - doffset)/dperiod) - (t - doffset)/dperiod) < 1e-8
  sys = x(1);               % 离散系统是一延迟环节
else
  sys = [  ];
```

```
function sys=mdlOutputs(t,x,u,doffset,dperiod)
if abs(round((t - doffset)/dperiod) - (t - doffset)/dperiod) <
    1e-8
  sys = x(2);             % 输出采样延迟
else
  sys = [  ];
end
```



9.4 编写C MEX S-函数

用C语言编写的S-函数具有以下优点：

(1) 执行速度快。

(2) 实时代码生成。

(3) 包含已有的C代码。

(4) 能够访问操作系统接口。

(5) 可以编写设备驱动。

下面介绍一些用C语言编写S-函数所需的几个基本概念。

9.4.1 MEX 文件

对于M文件S-函数，在MATLAB环境下可以通过解释器直接执行，对于C文件或其它语言编写的S-函数，则需要先编译成可以在MATLAB内运行的二进制代码：动态链接库或者静态库，然后才能使用，这些经过编译的二进制文件即是所谓的MEX文件，在Windows系统下MEX文件后缀为dll。要将C文件S-函数编译成动态库，需在MATLAB命令行下输入：

```
>> mex my_sfunction.c
```

要使用mex命令，首先需要在系统中安装一个C编译器。如果还没有设置编译器，则要在命令行下输入：

```
>> mex –setup
```

然后按照提示选取VC、BC或者其它的C编译器。推荐使用VC编译器。生成的文件my_sfunction.dll即是我们需要的动态库文件。当C文件中使用到其它库文件时，编译时应该在其后加上所需库文件名。

9.4.2 Simstruct 数据结构

一个称为SimStruct的数据结构描述了S-函数中所包含的系统。此结构在头文件simstruc.h 中定义。SimStruct将描述系统的所有信息，即封装系统的所有动态信息。它保存了指向系统的输入、状态、时间等存储区的指针，另外它还包含指向不同S-函数方法（S-函数例程）的指针。实际上整个Simulink框图模型本身也是通过一个SimStruct数据结构来描述的，它可以被视为与Simulink框图模型等价的表达。

9.4.3 工作向量（Work Vector）

在仿真过程中不释放的内存区域称之为持续存储区（Persistent Memory Storage），为全局变量或局部静态变量分配的内存就是这样的区域。当一个模型中出现同一个S-函数的多个实例时，这些全局变量或者局部静态变量就会发生冲突，导致仿真不能正确进行。因为这些实例使用了共同的动态链接库（MEX文件），正如在Windows下多个实例在内存中只有一个映像一样。此时Simulink为用户提供了工作向量（work vector）来解决这个问题。工作向量是Simulink为每个S-函数实例分配的持续存储区，它完全可以替代全局变量和局部静态变量。

表9.2 工作向量函数

函数名称	功 能 描 述	所 在 例 程
ssSetNumRWork	设置为实数型工作向量维数	在mdlInitializeSizes例程中分配，在mdlStart或者mdlInitializeConditions中初始化工作向量
ssSetNumIWork	为整型工作向量维数	
ssSetNumPWork	为指针数据类型工作向量维数	
ssSetRWorkvalue	设置实数型工作向量值	mdlOutputs
ssSetIWorkvalue	设置整型工作向量值	
ssSetPWorkvalue	设置指针型工作向量值	
ssGetRWorkvalue	读取实数型工作向量值	mdlUpdate或者mdlderivatives
ssGetIWorkvalue	读取整型工作向量值	
ssGetPWorkvalue	读取指针型工作向量值	

下面说明如何使用工作向量来保存和使用一个指向文件的指针。

(1) 首先在mdlInitializeSizes例程中设置指针工作向量的维数：

```
ssSetNumPWork(S, 1)      /*设置指针工作向量维数为1*/
```

(2) 在mdlStart例程中为该指针向量赋值：

```
static void mdlStart(real_T *x0, SimStruct *S)
{
FILE *fPtr;
void **PWork = ssGetPWork(S);
```

```
/*获取指向指针工作向量的指针，因为工作向量本身是指
    针数组，所以这里Pwork是指向指针
     的指针*/
fPtr = fopen("file.dat", "r");
PWork[0] = fPtr;  /*将文件指针存入指针工作向量 */
/* ssSetPWorkValue(S,0,fPtr); 显然更为简洁*/
}
```

(3) 在仿真结束时释放文件指针：

```
static void mdlTerminate(SimStruct *S)
{
if (ssGetPWork(S) != NULL) { /*首先判断是否存在指针工
    作向量*/
FILE *fPtr;
fPtr = (FILE *) ssGetPWorkValue(S,0);        /*再判断文件指
    针是否为空*/
if (fPtr != NULL)   { fclose(fPtr);  /*关闭文件*/ }
ssSetPWorkValue(S,0,NULL);          /*指针工作向量置空*/
}
}
```

9.4.4 C MEX S-函数流程

了解Simulink如何与S-函数相互作用完成动态系统的仿真对用户编写S-函数是非常有帮助的。前面已经对此进行了介绍，不同的是C MEX S-函数的流程控制更为精细，数据I/O也更为丰富，但是这里还有一些前面没有涉及到的内容。

图9.12显示了S-函数的数据交换过程。

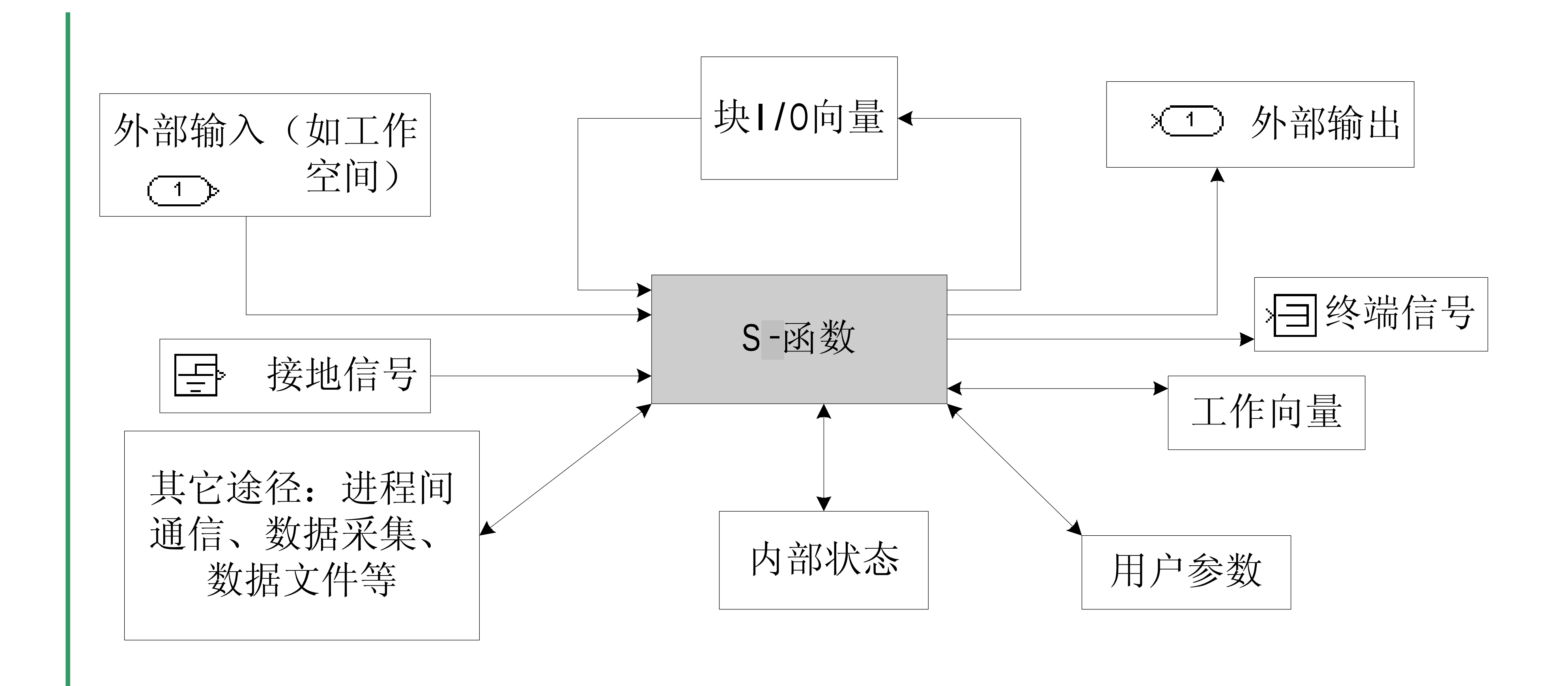


图9.12 S-函数的数据交换

除了这些输入输出数据外，S-函数还经常用到的内部数据有：

(1) 连续状态。

(2) 离散状态。

(3) 状态导数。

(4) 工作向量。

访问这些数据首先需要一套宏函数获取指向存储它们的存储器的指针，然后通过指针来访问。这时就需要特别注意指针的越界访问问题。

图9.13所示为Simulink引擎调用S-函数回调函数的过程。图中并没有列出全部的S-函数回调函数，尤其是初始化阶段远比图中所示要复杂。几个必需的例程在图中用实线表示，其它的用虚线表示。

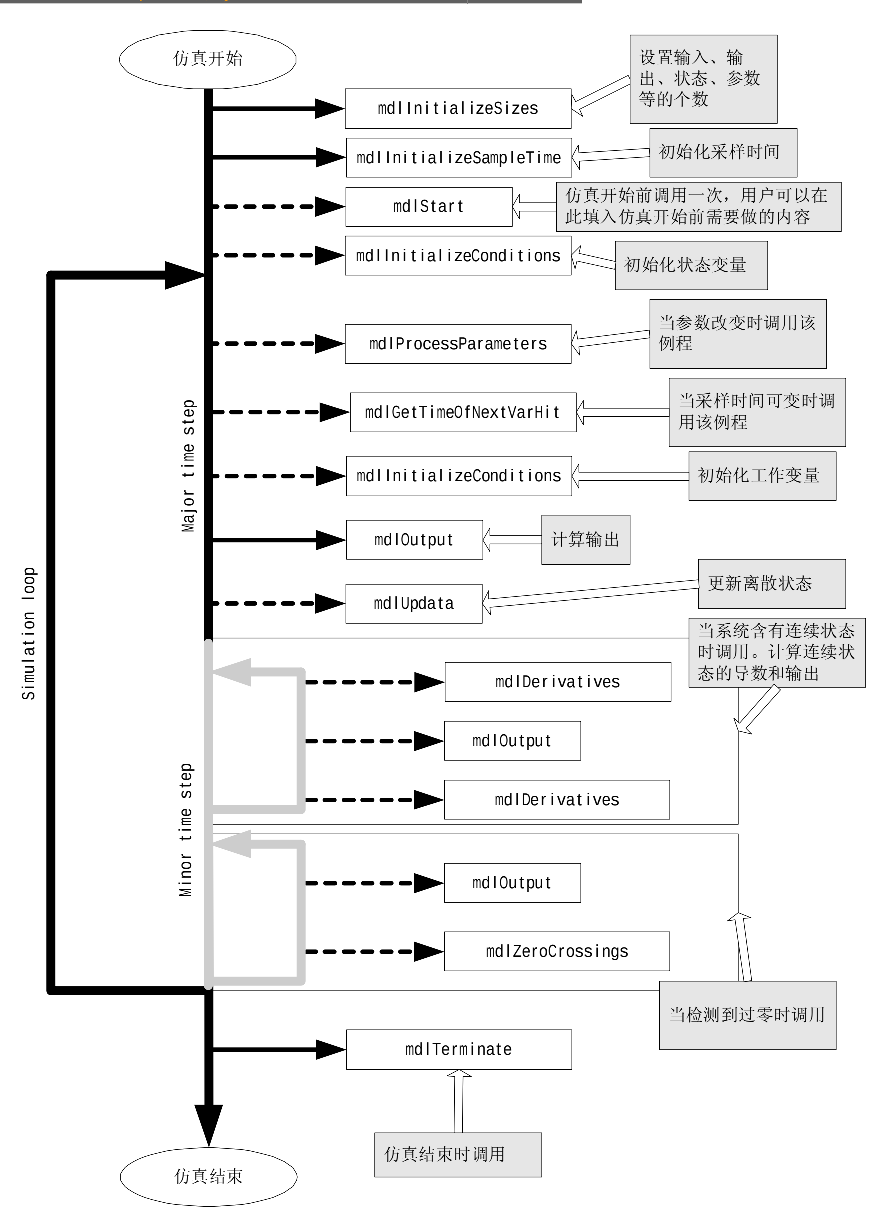


图9.13 C MEX S-函数工作流程

9.4.5 C MEX S-函数模板

与编写M文件S-函数一样，Simulink同样为用户提供了编写C MEX S-函数所需的模该文件只包含了常用的几个例程，这对于一般的应用已经足够了。文件sfuntmpl_doc.c则包含了所有的例程，并附有详细的注释。

每个C MEX S-函数的开头应包含下列语句：

```
#define S_FUNCTION_NAME your_sfunction_name_here
#define SFUNCTION_LEVEL 2
#include "simstruc.h"
```

另外在文件的顶部还应包含适当的头文件或定义其它的宏或者变量，就和编写普通的C程序一样。在C MEX S-函数的尾部必然包含下面几行代码：

```
#ifdef MATLAB_MEX_FILE
#include "simulink.c"
#else
#include "cg_sfun.h"
#endif
```

1. 初始化

C MEX S-函数的初始化部分包含下面三个不同的例程函数：

(1) mdlInitializeSizes：在该函数中给出各种数量信息。

(2) mdlInitializeSampleTimes：在该函数中给出采样时间。

(3) mdlInitializeConditions：在该函数中给出初始状态。

mdlInitializeSizes通过宏函数对状态、输入、输出等进行设置。工作向量的维数也是在mdlInitializeSizes中确定的。

表9.3 S-函数初始化所需宏函数

<table>
<tr><th>宏 函 数 定 义</th><th>功 能 描 述</th></tr>
<tr><td>ssSetNumContStates(S, numContStates)</td><td>设置连续状态个数</td></tr>
<tr><td>ssSetNumDiscStates(S, numDiscStates)</td><td>设置离散状态个数</td></tr>
<tr><td>ssSetNumOutputs(S, numOutputs)</td><td>设置输出个数</td></tr>
<tr><td>ssSetNumInputs(S, numInputs)</td><td>设置输入个数</td></tr>
<tr><td>sSetDirectFeedthrough(S, dirFeedThru)</td><td>设置是否存在直接前馈</td></tr>
<tr><td>ssSetNumSampleTimes(S, numSamplesTimes)</td><td>设置采样时间的数目</td></tr>
<tr><td>ssSetNumInputArgs(S, numInputArgs)</td><td>设置输入参数个数</td></tr>
<tr><td></td><td rowspan="3">设置各种工作向量的维数，实际上是为各个工作向量分配内存提供依据。见表9.7</td></tr>
<tr><td>ssSetNumRWork(S,numIWork)</td></tr>
<tr><td>ssSetNumPWork(S,numIWork)</td></tr>
</table>

2. 用输入和输出

在C MEX S-函数中，同样可以通过描述该S-函数的SimStruct数据结构对输入输出进行处理。在C MEX S-函数中，当需要对一个输入进行处理时，使用宏：

input=ssGetInputPortRealSignalPtrs(S,index)

返回值中含有指向输入向量的指针，其中的每个元素通过*input[i]来访问。指向输出向量的指针通过宏函数output=ssGetOutputPortRealSignal(S,index)得到。

如果想知道输出信号的宽度，使用宏：width=ssGetOutputPortWidth(S,index)；如果需要获得指向输入值的指针，使用宏：ssGetInputPortRealSignalPtrs(S,input_index)；如果需要获得指向输出值的指针，使用宏：ssGetOutputReaSignal(S, output_index)。表9.4列出了输入输出相关宏函数。

表9.4 输入输出相关宏函数

宏 函 数	功 能 描 述
ssGetInputPortRealSignalPtrs	获得指向输入的指针（double 类型）
ssGetInputPortSignalPtrs	获得指向输入的指针（其它数据类型）
ssGetInputPortWidth	获得指向输入信号宽度
ssGetInputPortOffsetTime	获得输入端口的采样时间偏移量
ssGetInputPortSampleTime	获得输入端口的采样时间
ssGetOutputPortRealSignal	获得指向输出的指针
ssGetOutputPortWidth	获得指向输出信号宽度
ssGetOutputPortOffsetTime	获得输出端口的采样时间偏移量
ssGetOutputPortSampleTime	获得输出端口的采样时间

例如：下面的一段代码获得指向输入和输出的指针，然后把输入乘以5后送往输出。

```
static void mdlOutputs(SimStruct *S, int_T tid)
{
    int_T              i;
    InputRealPtrsType uPtrs = ssGetInputPortRealSignalPtrs(S,0);
    real_T             *y = ssGetOutputPortRealSignal(S,0);
    int_T              width = ssGetOutputPortWidth(S,0);
    for (i=0; i<width; i++) {
        *y++ = 5 *(*uPtrs[i]);
    }
}
```

3. 使用参数

使用用户自定义参数时，在初始化中必须说明参数的个数。为了得到指向存储参数的数据结构的指针，使用宏：ptr=ssGetSFcnParam(S, index)；为了得到存储在这个数据结构中指向参数值本身的指针，使用宏：mxGetPr(ptr)；使用参数值时使用宏：param_value=*mxGetPr(ptr)。下面的代码首先获取参数gain的值，然后输入乘以gain作为输出：

```
#define GAIN=ssGetSFcnParam(S,0)
static void mdlOutputs(SimStruct *S, int_T tid)
{
    int_T           i;
    InputRealPtrsType  uPtrs = ssGetInputPortRealSignalPtrs(S,0);
    real_T           *y  = ssGetOutputPortRealSignal(S,0);
    int_T           width = ssGetOutputPortWidth(S,0);
    real_T           gain = *mxGetPr(GAIN);
for (i=0; i<width; i++) {
        *y++ = gain *(*uPtrs[i]);
    }
}
```

4. 使用状态

如果S-函数包含连续的或离散的状态，则需要编写mdlDerivatives或mdlUpdate子函数。若要得到指向离散状态向量的指针，使用宏：ssGetRealDiscStates(S)；若要得到指向连续状态向量的指针，使用宏：ssGetContStates(S)；在mdlDerivatives中，连续状态的导数应当通过状态和输入计算得到，并将SimStruct结构体中的状态导数指针指向得到的结果，这通过下面的宏完成：*dx=ssGetdX(S)，然后修改dx所指向的值。在多状态的情形下，通过索引得到dx中的单个元素。它们被返回给求解器通过积分求得状态。需要注意的是，在离散系统中，没有对应于dx的变量，由于状态是由S-函数来更新的，不需要求解器做任何额外的工作。

下面的一段代码描述了连续状态方程：

```
static void mdlDerivatives(SimStruct *S)
  {
          real_T            alpha = .01;
          real_T            beta = .02;
          real_T          *dx = ssGetdX(S);
```

```
    dx[0] = (1 - alpha*x[1])*x[0];
        dx[1] = (-1 + beta*x[0])*x[1];
  }
```

下面的一段代码描述了离散状态方程：

```
static void mdlUpdate(SimStruct *S, int_T tid)
{
```

```
real_T tempX[2] = {0.0, 0.0};
real_T *x = ssGetRealDiscStates(S);
InputRealPtrsType uPtrs = ssGetInputPortRealSignalPtrs(S,0);
UNUSED_ARG(tid); /* not used in single tasking mode */
tempX[0]= (1 - alpha*x[1])*x[0];;
tempX[1]= (-1 + beta*x[0])*x[1];
x[0]=tempX[0];
x[1]=tempX[1];
}
```

【例9.5】 S-函数csfunc描述了一个用状态方程表示的线性连续系统：并通过共享内存与其它程序交换数据。下面是该系统C MEX S-函数的完整源代码和注释。

```
#define S_FUNCTION_NAME csfunc  /* S-函数名 */
#define S_FUNCTION_LEVEL 2
#include "simstruc.h"
#include "windows.h"            /*创建共享内存所需头文件*/
#define U(element) (*uPtrs[element])          /* 宏定义，方便对
    输入的索引*/
```

```
/* 定义状态方程A B C D阵*/
static real_T A[2][2]={ { -0.09, -0.01 } ,
    {  1  ,  0   }     /*在C S-函数中有一套自己的数据*/
};                    /*类型表示方法, real_T表示双精  */
static real_T B[2][2]={ {  1   , -7    } ,        /*度,int_T 表示整型。通用的
    C*/
                      {  0   , -2    }   /* 语言数据类型标识同样适用*/
                     };
static real_T C[2][2]={ {  0   , 2     } ,
                      {  1   , -5    }
```

```
};
static real_T D[2][2]={ { -3  , 0   } ,  /*注意，存在馈通*/
                 { 1  , 0   }
               };
double *psharedmem;  /* 指向共享内存存储区的指针 */
HANDLE hfilemap;   /* 共享内存句柄 */
static void mdlInitializeSizes(SimStruct *S)
{
   ssSetNumSFcnParams(S, 0);  /* 不含用户参数，所以参数个数设为0
    */
```

```
if (ssGetNumSFcnParams(S) != ssGetSFcnParamsCount(S)) {
    return; /* Parameter mismatch will be reported by Simulink */
  }

  ssSetNumContStates(S, 2);    /*系统有两个连续状态*/
  ssSetNumDiscStates(S, 0);    /*系统没有离散状态*/
if (!ssSetNumInputPorts(S, 1)) return; /* 如果设置输入端口数为1失败，返回 */
/*S-functions 块只有一个输入端口，当需要多个输入时，使用mux模块把需要输入的信号集中
/*成一个向量*/
```

```
ssSetInputPortWidth(S, 0, 2);        /*输入信号宽度为2*/
    ssSetInputPortDirectFeedThrough(S, 0, 1);        /*设置馈通标志为
     1*/
    if (!ssSetNumOutputPorts(S, 1)) return;
    ssSetOutputPortWidth(S, 0, 2);        /*输出信号宽度为2*/
    ssSetNumSampleTimes(S, 1);    /*1个采样时间*/
    ssSetNumRWork(S, 0);        /*未使用工作向量 */
    ssSetNumIWork(S, 0);
    ssSetNumPWork(S, 0);
    ssSetNumModes(S, 0);
    ssSetNumNonsampledZCs(S, 0);
    ssSetOptions(S, SS_OPTION_EXCEPTION_FREE_CODE);
}
```

```
static void mdlInitializeSampleTimes(SimStruct *S)
{
ssSetSampleTime(S, 0, CONTINUOUS_SAMPLE_TIME);
/* 连续系统设采样时间为0，等同于 ssSetSampleTime(S, 0, 0);*/
    ssSetOffsetTime(S, 0, 0.0); /*偏移量为0*/
}

#define MDL_INITIALIZE_CONDITIONS
static void mdlInitializeConditions(SimStruct *S)
{
```

```
real_T *x0 = ssGetContStates(S);    /*获得指向连续状态的指针*/
   int_T  lp;
   for (lp=0;lp<2;lp++) {
      *x0++=0.0;                  /*各状态初始化为0*/
   }
   /*创建共享内存 */
   hfilemap=OpenFileMapping(
                        FILE_MAP_WRITE        ,
                        false,
                       "sharedmem"
                        );
```

```
/* 获得指向共享内存的指针*/
psharedmem=(double*)MapViewOfFile(hfilemap,FILE_MAP_WRITE,0,
    0,2*sizeof(double));
}

static void mdlOutputs(SimStruct *S, int_T tid)
{  /* 获得指向输出向量、连续状态向量和输入端口的的指针*/
    real_T          *y    = ssGetOutputPortRealSignal(S,0);
    real_T          *x    = ssGetContStates(S);
    InputRealPtrsType uPtrs = ssGetInputPortRealSignalPtrs(S,0);
     UNUSED_ARG(tid);
```

```
/* y=Cx+Du */   /* 输出方程*/
    y[0]=C[0][0]*x[0]+C[0][1]*x[1]+D[0][0]*U(0)+D[0][1]*U(1);
    y[1]=C[1][0]*x[0]+C[1][1]*x[1]+D[1][0]*U(0)+D[1][1]*U(1);
    psharedmem[0]=y[0]; /* 将输出放到共享内存中供其它进程使用*/
    psharedmem[1]=y[1];
}
#define MDL_DERIVATIVES
static void mdlDerivatives(SimStruct *S)
{
    real_T          *dx   = ssGetdX(S); /*获得指向状态导数向量的指针*/
    real_T          *x    = ssGetContStates(S);
    InputRealPtrsType uPtrs = ssGetInputPortRealSignalPtrs(S,0);
```

```
/* xdot=Ax+Bu */ /*连续状态方程 */
    dx[0]=A[0][0]*x[0]+A[0][1]*x[1]+B[0][0]*U(0)+B[0][1]*U(1);
    dx[1]=A[1][0]*x[0]+A[1][1]*x[1]+B[1][0]*U(0)+B[1][1]*U(1);
}

static void mdlTerminate(SimStruct *S)
{
    UNUSED_ARG(S);
    UnmapViewOfFile(psharedmem); /*在仿真结束时，释放共享内存 */
}
```

```
#ifdef  MATLAB_MEX_FILE    /* 是否编译成MEX文件 */
#include "simulink.c"            /* 其中包含MEX文件的接口
    方法 */
#else
#include "cg_sfun.h"          /* 代码生成注册函数 */
#endif
```

由以上程序可以看出，C MEX S-函数是通过一套宏函数获得指向存储在SimStruct中的输入、输出、状态、状态导数向量的指针来引用输入输出状态等变量的，从而完成对系统的描述。

9.4.6 S-函数包装程序

当用户需要将已有的程序、算法集成到Simulink框图模型中时，通常使用S-函数包装程序（MEX S-function Wrappers）来完成这个任务。所谓的S-函数包装程序就是一个可以调用其它模块代码的S-函数，实际上就是通过S-函数的形式来调用其它语言（MATLAB语言除外）编写的程序。使用外部模块时，需要在S-函数中将已有的代码声明为extern（即外部函数），并且还必须在mdlOutputs例程中调用这些已有的代码。

下面是一个简单的例子如下。C源文件：

```
/*wrapfcn.c*/
double wrapfcn(double input)
{
    return(input * 2.0);
}
```

C MEX S-函数源文件如下：

```
#define S_FUNCTION_NAME wrapsfcn
#define S_FUNCTION_LEVEL 2
#include “simstruc.h”
extern real_T wrapfcn( (real_T u); /*声明函数为外部函数*/
. . .
static void mdlOutputs(SimStruct *S, int_T tid)
```

```
{
InputRealPtrsType uPtrs = ssGetInputPortRealSignalPtrs(S,0);
real_T *y = ssGetOutputPortRealSignal(S,0);
*y = wrapfcn (*uPtrs[0]);  /*在mdlOutputs例程中调用外部
    函数wrapfcn*/
}
```

编译带有外部函数的S-函数时，只需将已有程序的源文件加在S-函数源文件的后面即可，如下所示：

```
>> mex wrapsfcn.c wrapfcn.c
```

9.4.7 S-function Builder

Simulink为用户编写常用的C MEX S-函数提供了方便的开发工具S-function Builder。使得用户无需了解众多的宏函数就可以编写出自己的S-函数，只要在对应的位置填入所需的信息和代码，S-function Builder就会自动生成C MEX S-函数源文件，并且编译起来也非常方便，只需单击Build按钮，就会生成用户所需要的MEX文件。

在Simulink库浏览器中双击S-function Builder块图标，即可打开如图9.14所示的S-function Builder界面。很容易看出1、3、4、5选项卡对应着S-函数的四个最常用的例程。打开S-function Builder为用户生成的C源文件，就会发现在各个页面填入的信息和代码被放入了对应的例程中。下面给出用户使用S-function Builder编写S-函数的步骤。

(1) 首先在S-function name编辑栏里填入S-函数名。

(2) 如果存在用户参数，在S-function parameters栏填入用户参数缺省值。

(3) 在图9.14所示的S-function Builder的Initialization页中按照提示填入仿真相关信息。

(4) 在Libraries选项卡中填入所需要的库文件（包括目录）、要包含的头文件，以及外部函数声明。如图9.15所示。

(5) 在Outputs、Continues Derivatives和Discrete Update页面填入输出方程、连续状态方程和离散状态方程，以及其它用户定制代码。

(6) 单击Build按钮，完成生成C代码、编译链接等工作。



S-Function Builder: simulink3/Functions & Tables/S-Function Builder

Parameters

S-function name: my_sfunction

S-function parameters: param1,param3,param

Build

1. Initialization | 2. Libraries | 3. Outputs | 4. Continuous Derivatives | 5. Discrete Update | 6. Build Info

Description

The S-Function Builder block creates a wrapper C-S-function with one input, one output and variable number of vector parameters. You can use this block to enter your own code or import legacy C code. This block also supports discrete and continuous states. The states must be of type real. Optionally the S-Function Builder block will generate a TLC file to be used with Real Time Workshop for code generation.

S-function settings

Input port width:	1	Number of discrete states:	2
Output port width:	1	Discrete states IC:	0,0
Number of parameters:	3	Number of continuous states	2
Sample time:	Inherited	Continuous states IC:	0,0
Discrete sample time value:			

Cancel　Help

图9.14 S-function Builder 初始化界面

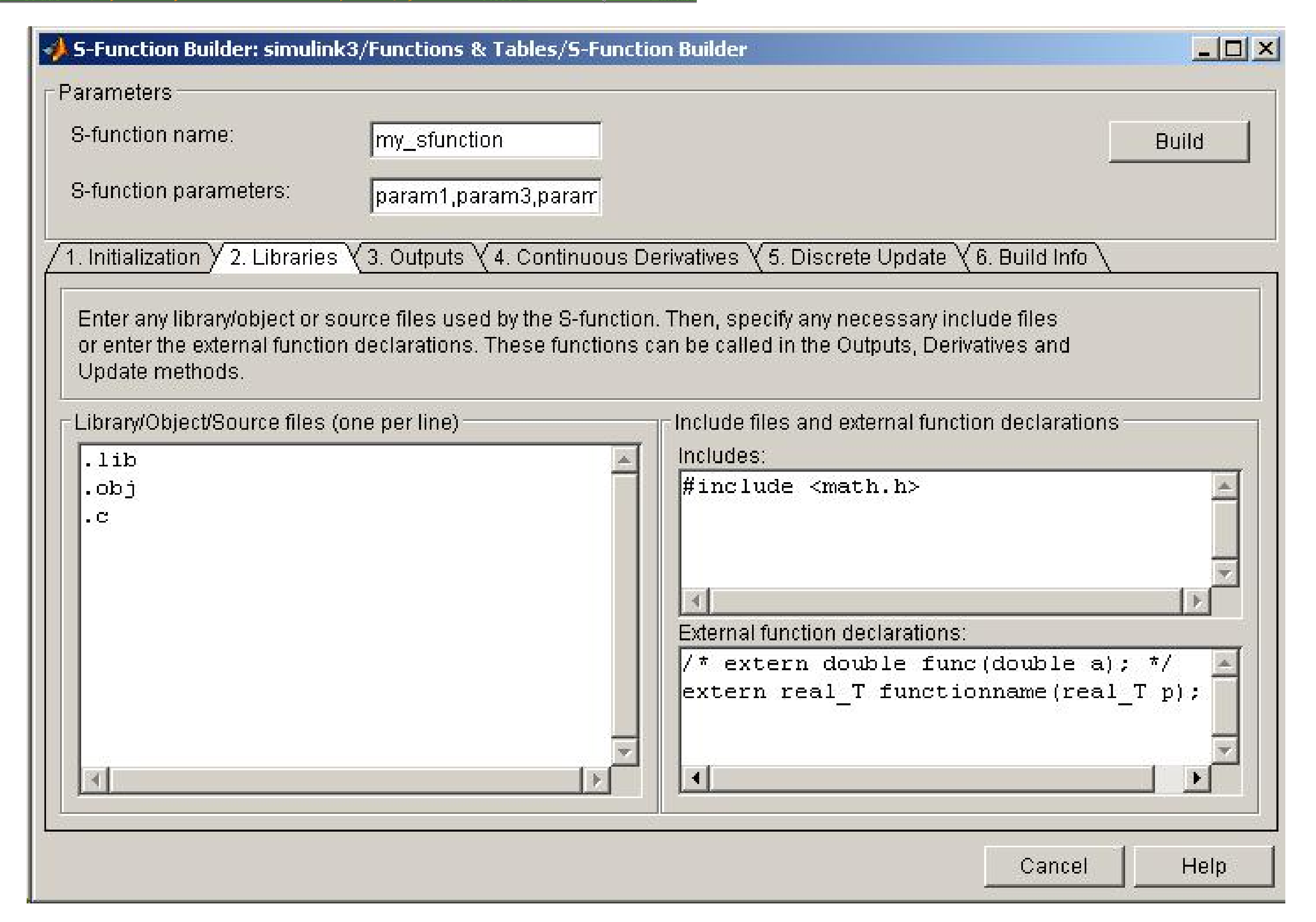


图9.15 S-function Builder库文件选项卡

第10章　控制系统设计分析



10.1 控制系统的线性分析

10.1.1滑艇动态方程及其线性化

1. 滑艇动力学方程

在滑艇的运行过程中，滑艇主要受到如下作用力的控制：滑艇自身的牵引力 $\boldsymbol{F}$ ，滑艇受到的水的阻力 $\boldsymbol{f}$ 。其中水的阻力 $f = v^2 - v$ ，为滑艇的运动速度。由运动学的相关定理可知，整个滑艇系统的动力学方程为

$$\dot{v} = \frac{1}{m}(F - (v^2 - v))$$

其中为滑艇的质量。由滑艇系统的动力学方程易知，此系统为一非线性系统。下面来建立此系统的Simulink模型并进行线性分析。

2. 滑艇速度控制系统的模型建立与仿真

使用下面的Simulink模块建立滑艇速度控制系统的模型：

(1) Sources模块库中的Step模块：用来产生滑艇的牵引力。

(2) Subsystems模块库中的Subsystem模块：构成滑艇速度控制器子系统。

(3) Sinks模块库中的Scope模块：输出滑艇的速度。

(4) Functions & Tables模块库中的Fcn模块：求取水的阻力。

(5) 其它模块：Math模块库中的Gain模块、Continuous模块库中的Integrator模块。

使用Simulink建立的系统模型框图如图10.1所示。

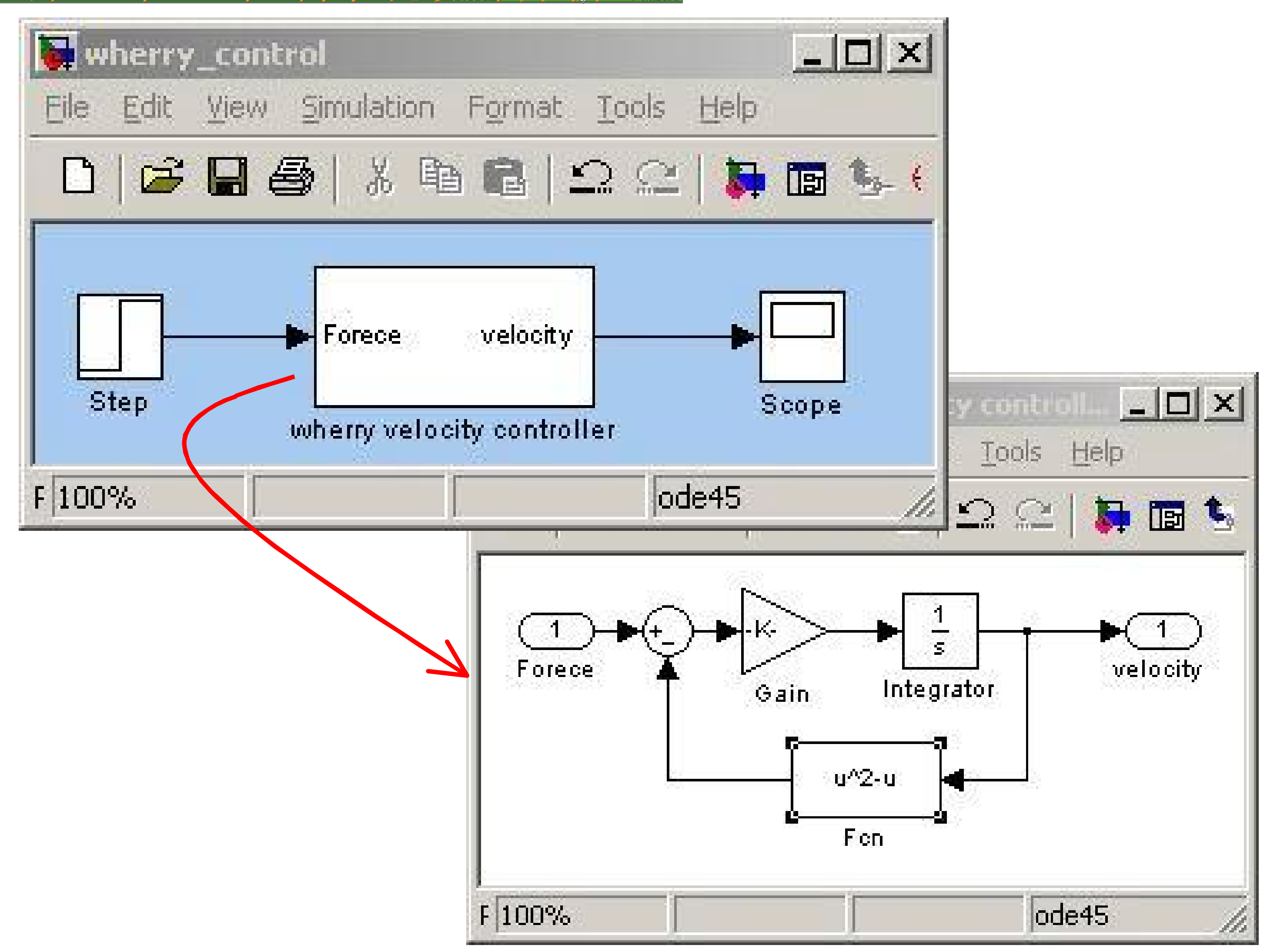
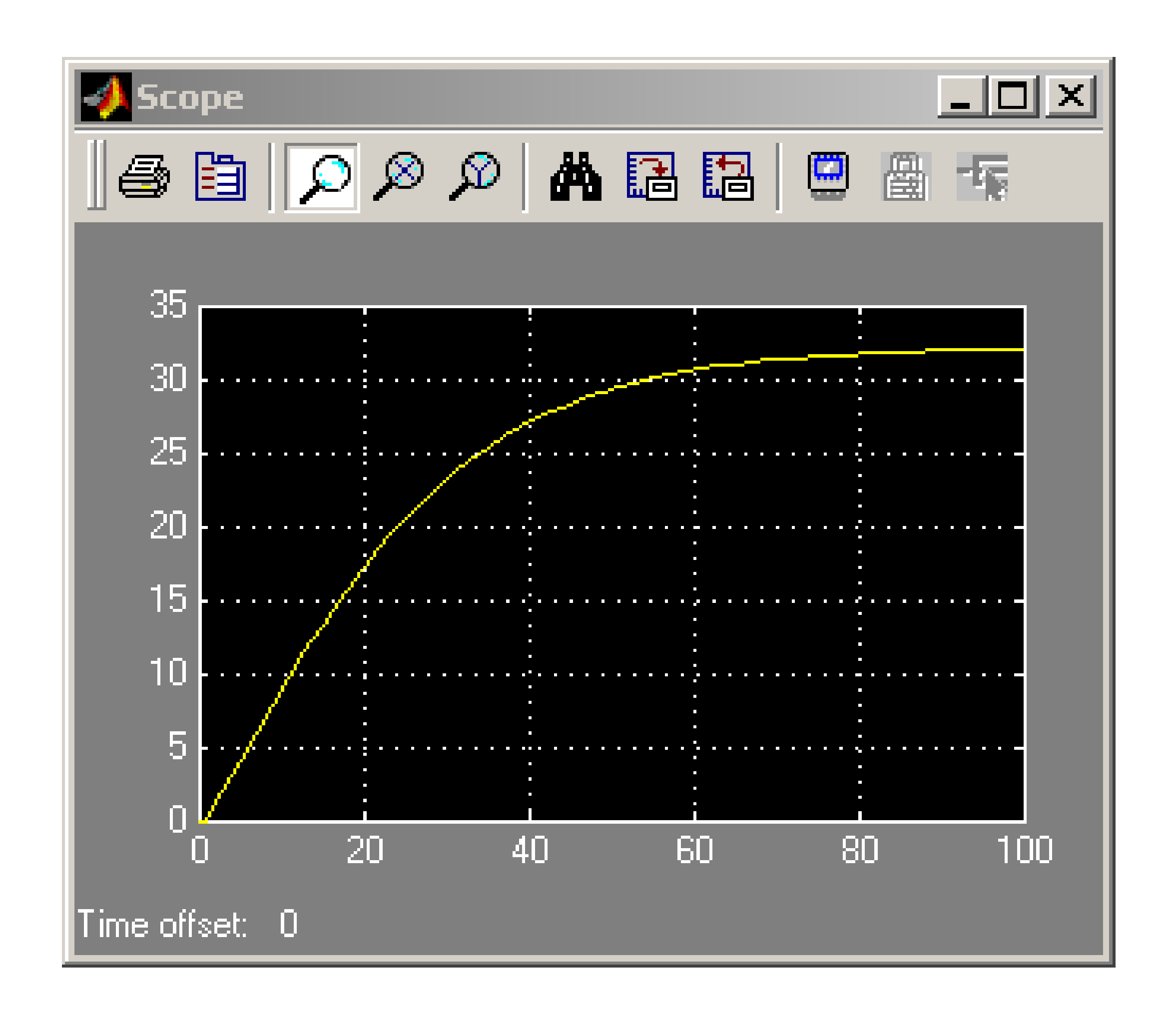


图10.1 滑艇速度控制系统模型框图

然后设置正确的系统模型参数与仿真参数对此系统进行仿真，其中Step的Final Value值设置为1000（即滑艇牵引力）、子系统中增益模块Gain的取值为1/1000（即1/*m*）、Fcn模块的expression设置为*u*^2-*u*（求取水的阻力）、系统仿真时间为0至100 s。图10.2为系统仿真的结果。



滑艇在牵引力（值1000）的作用下，速度在经过80 s左右的时间后，由0上升并稳定在33 km/h

图10.2 滑艇系统仿真结果

3. 滑艇速度控制器系统的线性化

对于滑艇速度控制器系统而言，如果要在比赛中获得胜利，则滑艇必须在尽可能短的时间内达到最大速度。设此速度控制器所能达到的最大速度为100 mph（miles per hour，英里每小时）。而在前面所提供的滑艇牵引力仅为1000，故需要设置合适的牵引力对速度控制器进行操纵。

既然滑艇速度最大值为100 mph，因此在对滑艇速度控制系统进行线性化时，希望此系统能够使滑艇的速度基本稳定在最大速度处。换句话说，系统的工作点应该选择为使速度达到100 mph时的系统输入与系统状态。由于对非线性系统进行线性化表示需要给出系统所在的操作点（即平衡点），因此在对滑艇速度控制系统进行线性化之前，需要获得滑艇速度稳定在100 mph处的系统平衡点。按照如下步骤可以获得滑艇速度控制系统的平衡点：

(1) 修改系统模型，如图10.3所示。

其中Inport、Outport分别表示系统的输入与输出，增益模块的作用是将速度单位km/h转变为mph，其值为5/8。

(2) 求取滑艇速度控制系统在此工作点处的平衡状态。

在MATLAB命令窗口中使用trim命令获得系统在输出为100 mph时的平衡状态：

```
>> [x, u, y, dx]=trim('wherry_control', [ ], [ ], 100, [ ], [ ], 1)
        x =160
        u =
```

```
 2.5440e+004

y =

   100

dx =

 -1.0914e-014
```

(3) 求取滑艇速度控制系统的线性系统描述。

在获得使滑艇速度稳定在100 mph处时系统的平衡点x、u与y之后，在MATLAB命令窗口中使用linmod命令便可以获得相应的线性系统描述，如下所示：

```
>> [A,B,C,D]=linmod('wherry_control',x,u)

A =

-0.3190

B =

1.0000e-003
```

C =

0.6250

D =

0

从而得到线性化后系统的状态空间描述，其中A、B、C与D是线性系统的状态空间矩阵。故相应的线性系统的状态空间描述方程为

$$\dot{x} = -0.319x + 0.001u$$
$$y = 0.625x$$

4. 使用LTI Viewer进行非线性系统的线性分析

除了使用前面的命令行方式对非线性系统进行线性化处理分析之外，Simulink还提供了友好的图形界面对非线性系统进行线性分析。使用图形界面可以使用户对非线性系统的性能有一个非常直观的认识与理解。

在滑艇控制系统模型wherry_control中，选择Tools菜单下的Linear Analysis命令。此时将打开LTI线性时不变系统浏览器窗口，如图10.4所示。

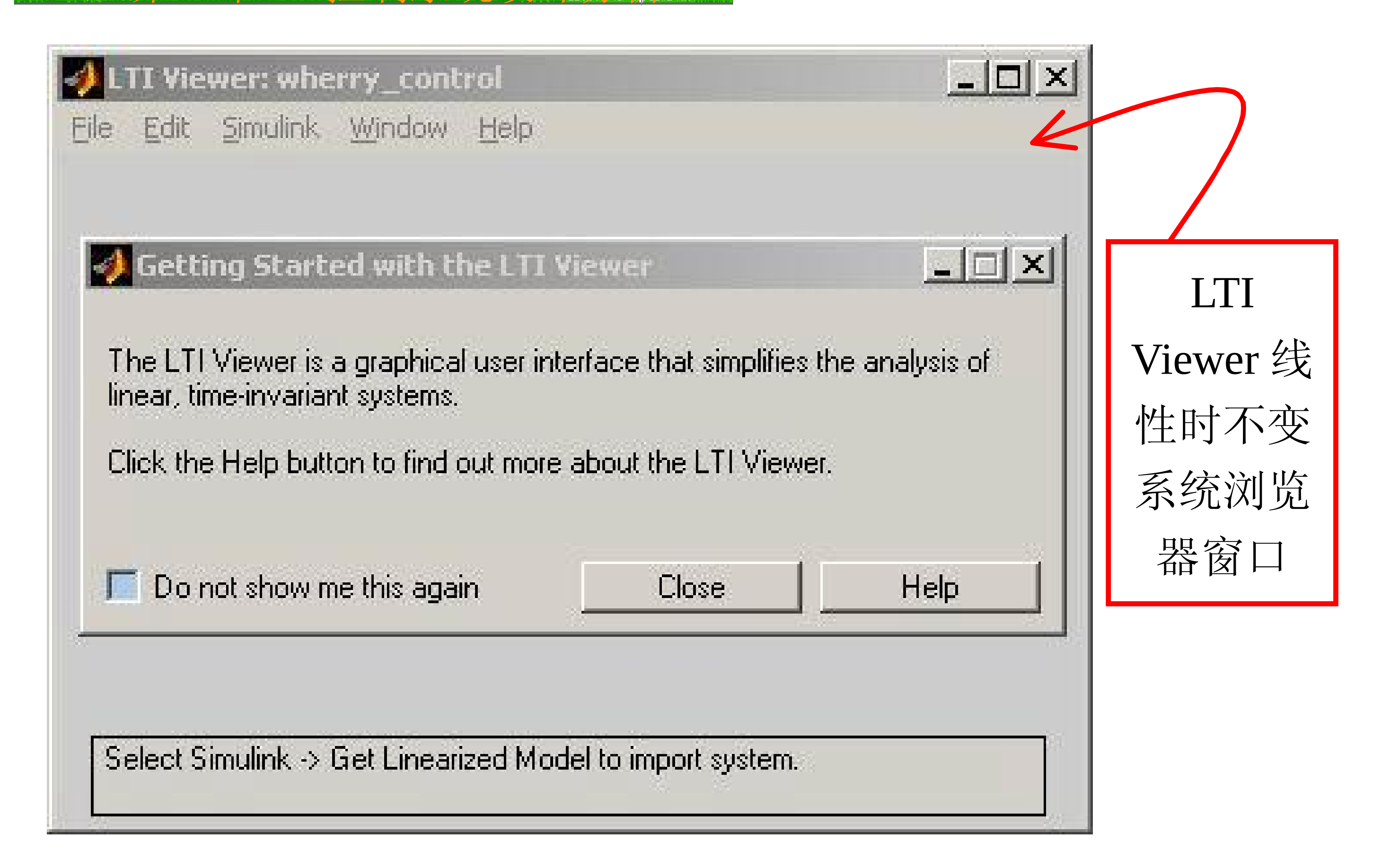
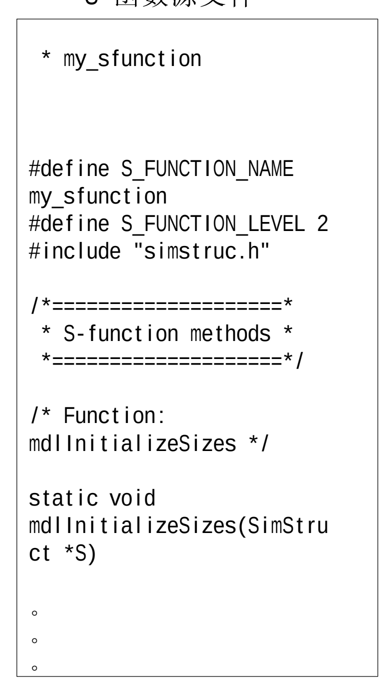


图10.4 LTI Viewer窗口界面

在对非线性系统进行线性分析时，用户必须指定所分析的非线性系统中的参考输入点（即系统分析输入点Input point）与参考输出点（即系统分析输出点Output point）；对于wherry_control滑艇速度控制系统而言，使用LTI Viewer对其进行线性分析的步骤如下：

(1) 在滑艇速度控制系统模型中加入Input point与Output point，如图10.5所示。

从Model_Inputs_and_Outputs模块库中将参考输入点与参考输出点拖动到滑艇速度控制系统模型中


S-函数源文件

```
 * my_sfunction


#define S_FUNCTION_NAME
my_sfunction
#define S_FUNCTION_LEVEL 2
#include "simstruc.h"

/*===================*
 * S-function methods *
 *===================*/

/* Function:
mdlInitializeSizes */

static void
mdlInitializeSizes(SimStru
ct *S)

。
。
。
```

图10.5 在系统模型中加入Input point与Output point

(2) 设置滑艇速度控制系统的操作点。

所谓的操作点便是在前面使用trim命令所获得的系统平衡点状态，即使滑艇速度能够稳定在100 mph的系统输入值与系统状态值。选择LTI Viewer窗口中Simulink菜单下的Set Operating Point命令设置系统操作点，如图10.6所示。

Operating Point: wherry_control

States

Linearize about

Initial states in Simulink diagram

Zero state values

User-defined state values

Names	Value
1. ...wherry velocity controller/Integrator	160

Inputs

Names	Value
In1	25440

Help OK Cancel

系统输入

系统状态

图10.6 设置系统操作点



图10.7 操作点改变提示对话框

(3) 获得滑艇速度控制系统在当前操作点的线性表示。

在对滑艇速度控制系统的操作点进行正确设置之后，单击OK按钮。此时将弹出如图10.7所示的对话框，以提示用户系统操作点的改变，选择Yes应用新的操作点设置。随后将在LTI Viewer窗口中显示滑艇速度控制系统的阶跃响应，如图10.8所示。

图10.8 滑艇速度控制系统的阶跃响应

10.1.2 线性时不变系统浏览器LTI Viewer介绍

1. 绘制系统的不同响应曲线

在默认的情况下，LTI Viewer绘制系统在单位阶跃信号输入下的系统响应曲线（即阶跃响应）。其实使用LTI Viewer可以绘制不同的系统响应，在LTI Viewer图形绘制窗口中单击鼠标右键，选择弹出菜单Plot Type下的子菜单，可以在LTI Viewer图形绘制窗口中绘制不同的系统响应曲线，如图10.9所示。

Plot Type ▸
Systems ▸
Characteristics ▸
Grid
Zoom ▸
Properties...

✔ Step
Impulse
Bode
Bode Mag.
Nyquist
Nichols
Sigma
Pole/Zero

阶跃响应
脉冲响应
波特图
波特图幅值
奈奎斯特图
尼科尔斯图
奇异值分析
零极点图

图10.9 系统响应曲线绘制选择

LTI Viewer: wherry_control

File Edit Simulink Window Help

Impulse Response

From: Input Point

x 10^-4

Amplitude

To: Output Point

Time (sec)

Plot type changed to Impulse.

选择 Plot Type 中的 Impulse 命令所绘制的系统单位脉冲响应曲线

图10.10 滑艇速度控制系统的单位脉冲响应曲线

2. 改变系统响应曲线绘制布局

在默认的情况下，LTI Viewer图形绘制窗口中仅仅绘制一个系统响应曲线。如果用户需configurations对LTI Viewer图形绘制窗口的布局进行改变，并在指定的位置绘制指定的响

应曲线。图10.11为响应曲线绘制布局设置对话框，以及采用图中给出的设置同时绘制6幅不同的响应曲线。

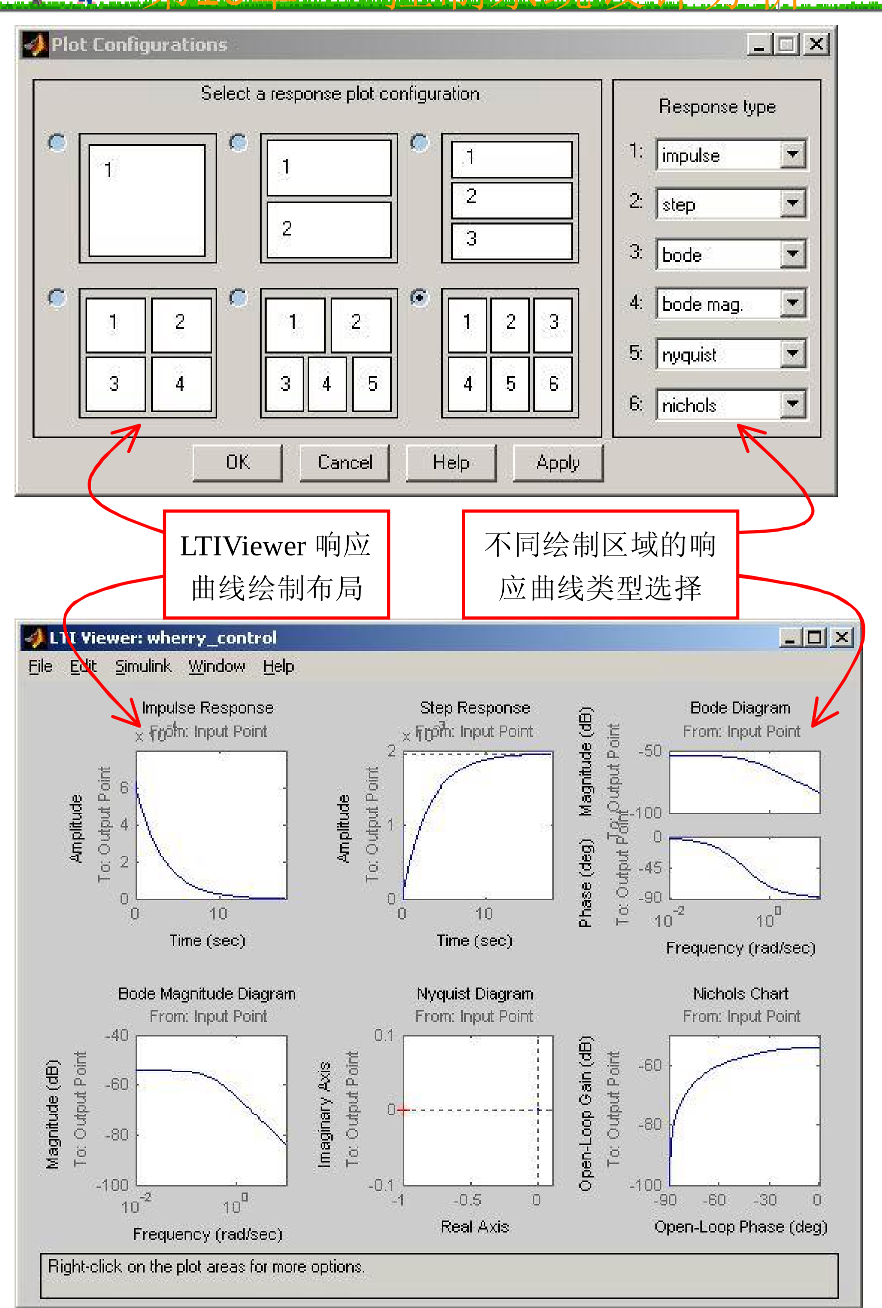


图10.11　响应曲线布局设置及绘制结果

3. 系统时域与频域性能分析

使用LTI Viewer不仅可以方便地绘制系统的各种响应曲线，还可以从系统响应曲线中获得系统响应信息，从而使用户可以对系统性能进行快速的分析。首先，通过单击系统响应曲线上任意一点，可以获得动态系统在此时刻的所有信息，包括运行系统的名称、系统的输入输出以及其它与此响应类型相匹配的系统性能参数。例如，对于滑艇速度控制系统的单位脉冲响应，单击响应曲线中的任意一点，可以获得系统响应曲线上此点所对应的系统运行时刻（Time）、系统输入值（Amplitude）等信息，如图10.12所示。

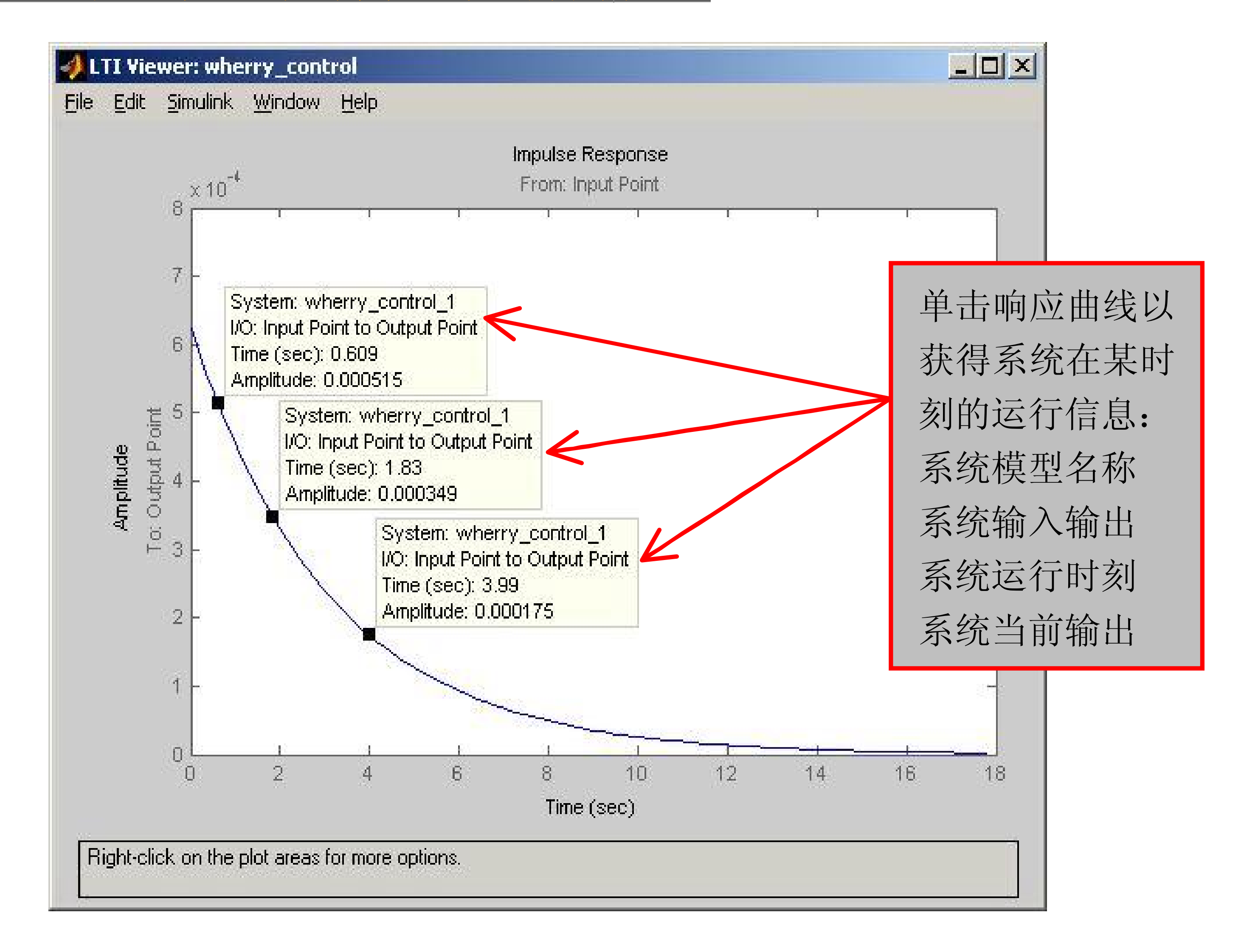


图10.12 从系统响应曲线获得系统运行信息

其次，用户可以在LTI Viewer图形绘制窗口中单击鼠标右键，使用右键弹出菜单中的Characteristics子菜单获得系统不同响应的特性参数，对于不同的系统响应类型，Characteristics菜单的内容并不相同。图10.13所示为阶跃响应的特性参数。

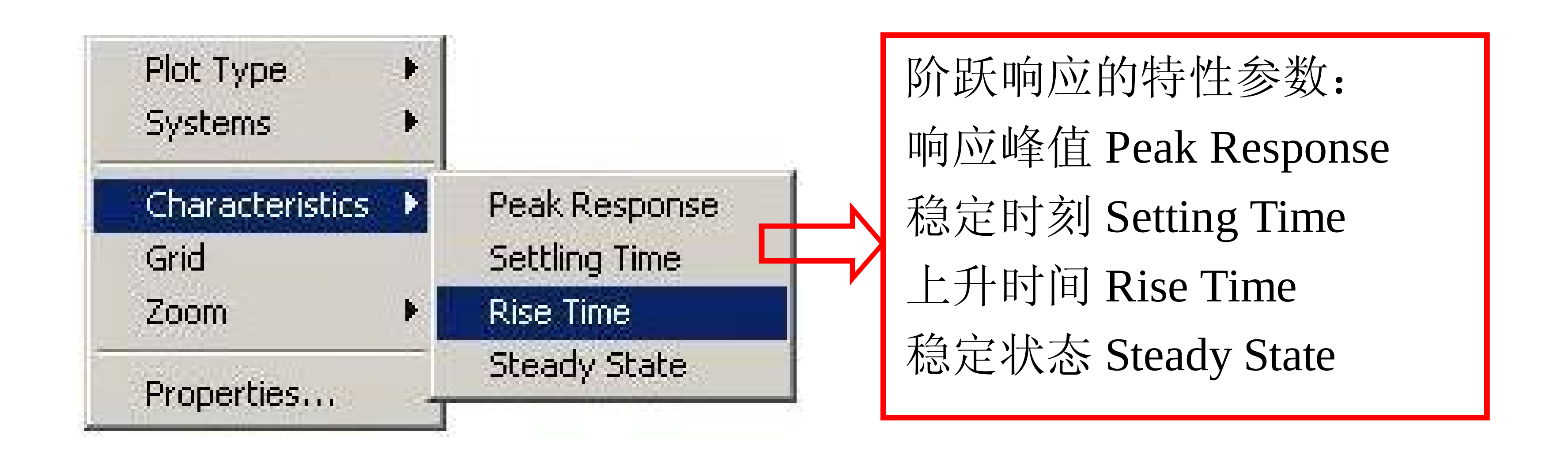


图10.13 阶跃响应的特性参数

选择Characteristics右键弹出菜单中的Rise Time可以获得系统阶跃响应的上升时间（即系统输出从终值的10%到终值的90%所需要的时间）。此时在LTI Viewer绘制的阶跃响应曲线中将出现上升时间标记点，单击此标记点即可获得上升时间，如图10.14所示。

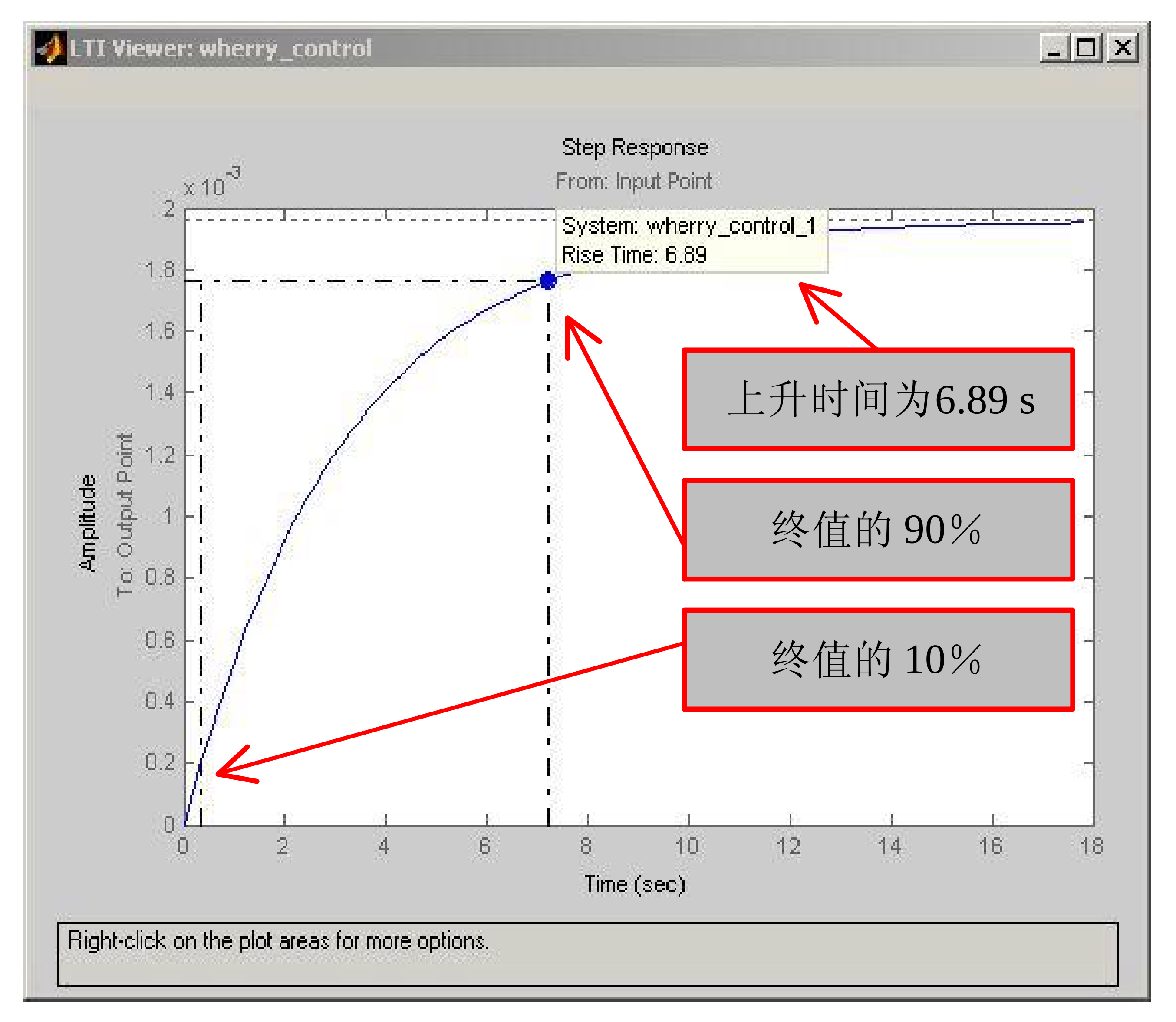


图10.14 阶跃响应的上升时间

4. LTI Viewer图形界面的高级控制

前面简单介绍了LTI Viewer响应曲线绘制窗口的布局设置。Simulink最为突出的特点之一就是其强大的图形功能。在Simulink中，任何图形都是特定的对象，用户可以对其进行强有力的操作与控制。下面介绍如何对LTI Viewer图形窗口进行更为高级的控制。

对LTI Viewer图形窗口的控制有两种方式。

一是对整个浏览器窗口Viewer进行控制：单击LTI Viewer窗口的Edit菜单下的Viewer Preferences命令对浏览器进行设置（此设置的作用范围为LTI Viewer窗口以及所有系统响应曲线绘制区域）。在此对话框中共有4个选项卡：

(1) Units选项卡：设置图形显示时频率、幅值以及相位的单位。

(2) Style选项卡：设置图形显示时的字体、颜色以及绘图网格。

(3) Characteristics选项卡：设置系统响应曲线的特性参数。

(4) Parameters选项卡：设置系统响应输出的时间变量与频率变量。

二是对某一系统响应曲线绘制窗口进行操作：在系统响应曲线绘制窗口中单击鼠标右键，选择弹出菜单中的Properties对指定响应曲线的显示进行设置。此对话框中共有5个选项卡：

(1) Labels选项卡：设置系统响应曲线图形窗口的坐标轴名称、窗口名称。

(2) Limits选项卡：设置坐标轴的输出范围。

(3) Units选项卡：设置系统响应曲线图形窗口的显示单位。

(4) Style选项卡：设置系统响应曲线图形窗口的字体、颜色以及绘制网格。

(5) Characteristics选项卡：设置系统响应曲线的特性参数。

5. 线性化模型的输出

在使用LTI Viewer对非线性系统进行线性分析之后，用户可以使用File菜单下的Export命令将此线性化线性模型输出到MATLAB工作空间之中。此时将打开如图10.15所示的窗口。

图中列出了一系列系统模型的名称，选择其中的一个以输出其线性化模型。输出方式有两种：

(1) 将系统线性化模型输出至MATLAB工作空间中，此时输出的线性化模型为一LTI对象变量（LTI对象的相关知识将在下面的内容中给出）。

(2) 将系统线性化模型输出至磁盘文件（*.mat文件，即MATLAB数据文件）。

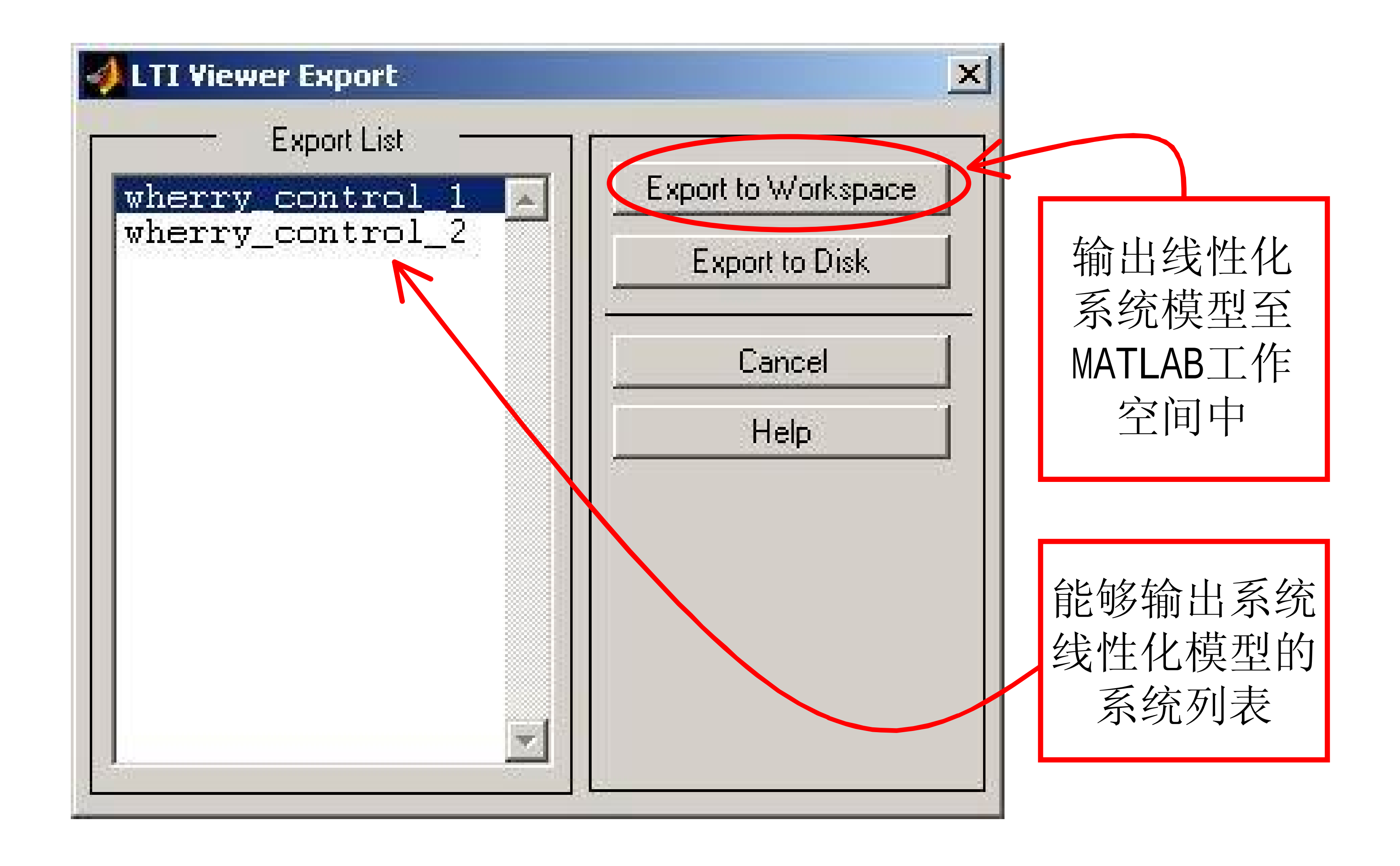


图10.15 输出线性化系统模型

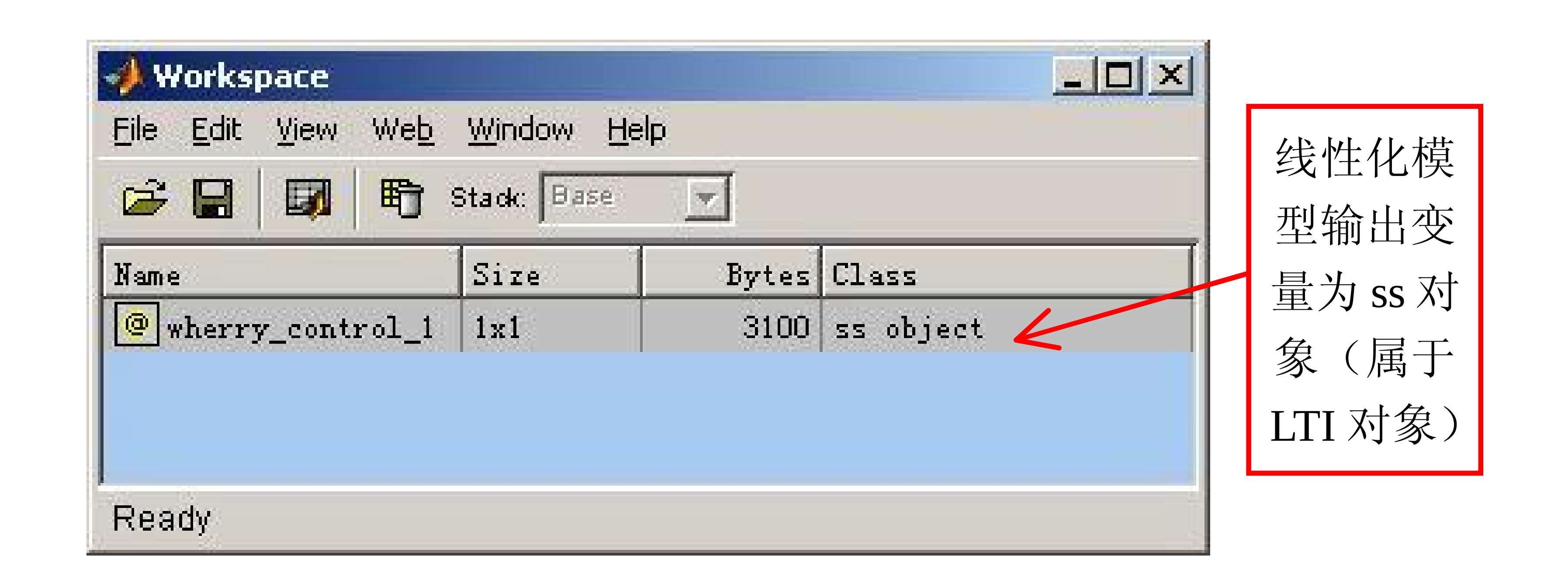


图10.16 线性化系统模型的输出变量

10.1.3 LTI线性时不变系统对象介绍

在使用LTI Viewer线性时不变系统浏览器对系统进行线性分析时，如果将线性化的系统模型输出至MATLAB的工作空间中，则会在MATLAB工作空间中出现类型为ss对象的变量（ss object）。ss对象是LTI对象（线性时不变系统对象）的一种形式，LTI对象是控制系统工具箱中最为基本的数据类型。此类型数据拥有描述一个线性时不变系统的所有信息，LTI对象有如下的三种方式：

(1)ss对象：封装了由状态空间模型描述的线性时不变系统的所有数据。

(2) tf对象：封装了由传递函数模型描述的线性时不变系统的所有数据。

(3) zpk对象：封装了由零极点模型描述的线性时不变系统的所有数据。

1. LTI对象的属性

不同的LTI对象除了拥有某些共同的属性之外，还有属于每一种对象本身的特殊属性。使用get命令可以获得LTI对象的所有属性。对于由LTI Viewer输出的描述滑艇速度控制系统的线性化模型的ss对象wherry_control_1而言，使用get命令获得ss对象的属性如下：

```
>>get(wherry_control_1)
        a: -0.319
        b: 0.001
        c: 0.625
        d: 0
        e: []
```

```
StateName: {[1x53 char]}
       Ts: 0
  ioDelay: 0
 InputDelay: 0
OutputDelay: 0
 InputName: {'Input Point'}
OutputName: {'Output Point'}
 InputGroup: {0x2 cell}
OutputGroup: {0x2 cell}
     Notes: {}
  UserData: [  ]
```

其中从Ts开始之后的属性为所有LTI对象均具有的属性，分别用来描述LTI系统的采样时间、输入输出延迟、输入输出端口名称以及其它用户自定义的数据等等。而在Ts之前的属性则属于不同对象本身所特有的，用来描述线性时不变系统，如wherry_control_1中的状态空间描述矩阵a、b、c、d、e以及系统状态变量。如需获得LTI对象中某个属性的取值，则可以使用与结构体类似的语法，如

```
>>state=wherry_control_1.StateName
state =
'wherry_control/wherry velocity controller/Integrator'
```

2. 对LTI对象的基本操作

由于LTI对象是控制工具箱中最基本的数据类型，因而MATLAB支持对LTI对象的直接操作。用户可以使用控制工具箱中的系统分析设计命令对这些LTI对象进行操作，而且由于LTI对象包括线性系统是连续还是离散的信息，因此可以使用同样的命令对连续系统与离散系统进行操作。这里并不打算对控制工具箱中的函数命令进行介绍，而仅介绍LTI对象本身的一些简单操作。

(1) 生成LTI对象。使用ss、tf及tpk可以建立不同类型的LTI对象，如使用tf命令建立使用传递函数描述的线性时不变系统对象。

```
>>mysys_tf=tf([2 1], [1 3 2])                    % 生成tf对象
mysys_tf
Transfer function:
        2 s + 1
   -------------------
      s^2 + 3 s + 2
```

(2) LTI对象间的相互转换。同样可以使用ss、tf及zpk进行LTI对象之间的相互转换，如

```
>>mysys_ss=ss(mysys_tf) %将tf对象转换为ss对象
      a =
   x1  x2
x1  –3  –1
   x2   2   0
 b =
u1
 x1   2
 x2   0
```

```
c =

        x1    x2

   y1      1  0.25

d =

      u1

   y1   0

Continuous-time model.          % 指明系统为连续时间系统
```

(3) 获得与设置LTI对象属性。在前面已经介绍，这里不再赘述。

(4) 线性时不变系统的并联，即LTI对象的相加，如

```
>>sys1=tf([2 1], [1 3 2]);        % 生成系统1
>>sys2=tf([1 1], [ 2 3 –1]);% 生成系统2
>>sys=sys1+sys2                    % 并联系统1与2
Transfer function:
  5 s^3 + 12 s^2 + 6 s + 1
-------------------------------------------
2 s^4 + 9 s^3 + 12 s^2 + 3 s – 2
```



10.2 线性控制系统设计分析

10.2.1 控制系统工具箱简介

控制系统工具箱是MATLAB中所提供的对控制系统进行辅助设计的功能强大的开发设计工具。它包含了丰富的线性系统分析和设计函数，并以LTI对象为基本数据类型对线性时不变系统进行操作与控制。控制系统工具箱能够完成系统的时域和频域分析。在控制系统工具箱中，可以使用不同的方法设计线性反馈系统，如

(1) 根轨迹设计分析法。

(2) 极点配置法。

(3) 和控制。

(4) 状态观测器设计。

(5) 规范型实现设计。

在使用控制系统工具箱完成线性反馈系统设计之后，便可以通过Simulink进行系统的动态仿真，从而得到真实的、非线性系统的响应，进一步对控制器进行验证。

10.2.2系统分析与设计简介

控制系统工具箱中最基本的数据类型为LTI对象。无论LTI对象的类型如何，都可以使用相同的命令对其进行分析，因为LTI对象包含了线性时不变系统的所有信息。这里简单介绍一下用来对由LTI对象所描述的线性时不变系统进行分析设计的命令函数。

1. 动态分析函数

动态分析函数有pole(sys)、dcgain(sys)、tzero(sys)、damp(sys)及norm(sys)等等。对于由如下命令：

```
>>mysys_tf=tf([2 1], [1 3 2]);
```

生成的LTI对象mysys_tf所描述的线性时不变系统，可以使用上述函数对其进行分析，

例如：

```
>>pole (mysys_tf)                    % 求取系统极点
ans =
   -2
   -1
>> dcgain(mysys_tf)  % 求取系统直流增益
ans =
0.5000
```

2. 时域与频域分析函数

时域与频域分析函数有step(sys)、bode(sys)、impulse(sys)、nichols(sys)、initial(sys,x0)、nyquist(sys)、lsim(sys,u,t)以及sigma(sys)等。例如：

>> step(mysys_tf)　　　　　% 绘制系统的单位阶跃响应曲线

>> figure，nyquist(mysys_tf)　　　　% 在新的图形窗口绘制系统的nyquist图

使用这两个命令分别绘制线性时不变系统mysys_tf的单位阶跃响应与nyquist图，与对LTI Viewer中系统响应曲线的操作相类似，用户可以使用右键弹出式菜单获得系统的时域（或频域）的动态响应（或动态性能），如图10.17所示。

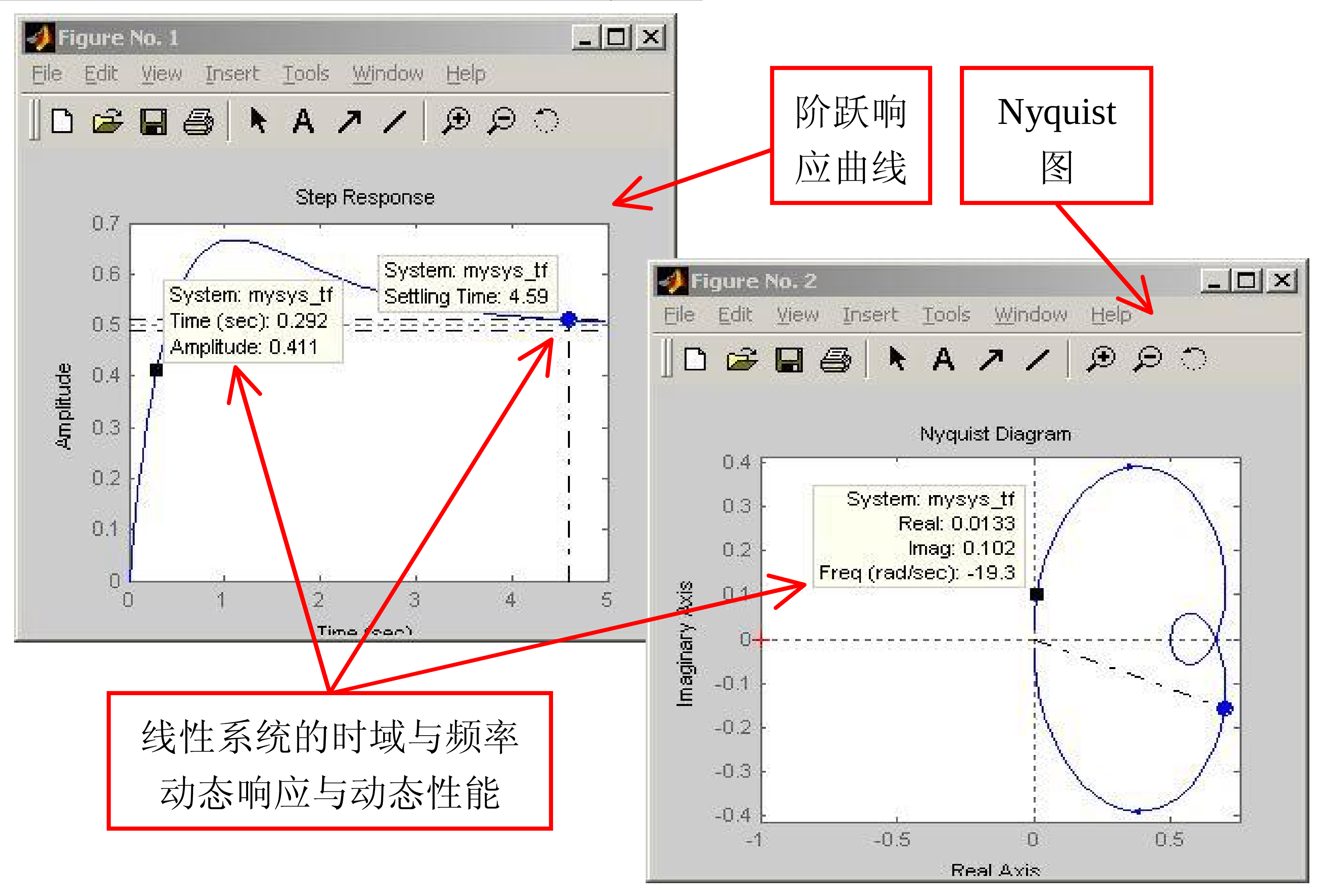


图10.17 线性时不变系统mysys_tf的阶跃响应曲线与Nyquist图

3. 补偿器设计

使用控制系统工具箱中的函数还可以进行各种系统的补偿设计，如LQG（Linear-Quadratic-Gaussian，线性二次型设计）、Root Locus（线性系统的根轨迹设计）、Pole placement（线性系统的极点配置）以及Observer-based regulator（线性系统观测器设计）等。由于这些内容涉及较多的知识，在此不作介绍。

10.2.3 单输入单输出系统设计工具

在对非线性系统的线性分析技术进行介绍时，线性时不变系统浏览器LTI Viewer是进行系统线性分析的最为直观的图形界面，使用LTI Viewer使得用户对系统的线性分析变得简单而直观。其实LTI Viewer只是控制系统工具箱中所提供的较为简单的工具，主要用来完成系统的分析与线性化处理，而并非系统设计。

1. 启动SISO设计器

在MATLAB命令窗口中键入如下的命令启动SISO设计器：

```
>>sisotool
```

启动后的SISO设计器如图10.18所示。

在默认的情况下，SISO设计器同时启用系统根轨迹编辑器与开环波特图编辑器，如图10.18所示。当然，此时尚未进行系统设计，故不显示根轨迹与开环波特图。

SISO Design Tool

File Edit View Compensators Tools Window Help

Current Compensator

C(s) =

Root Locus Editor (C)

Real Axis

Open-Loop Bode Editor (C)

Frequency (rad/sec)

Use Import Model... off the File menu to import the plant data.

补偿器设置

根轨迹编辑器

控制系统结构

波特图编辑器

图10.18 SISO设计器

2. 输入系统数据（Import System Data）

在启动SISO设计器之后，需要为所设计的线性系统输入数据，选择SISO设计器中File菜单下的Import命令输入系统数据，此时将打开如图10.19所示的对话框。

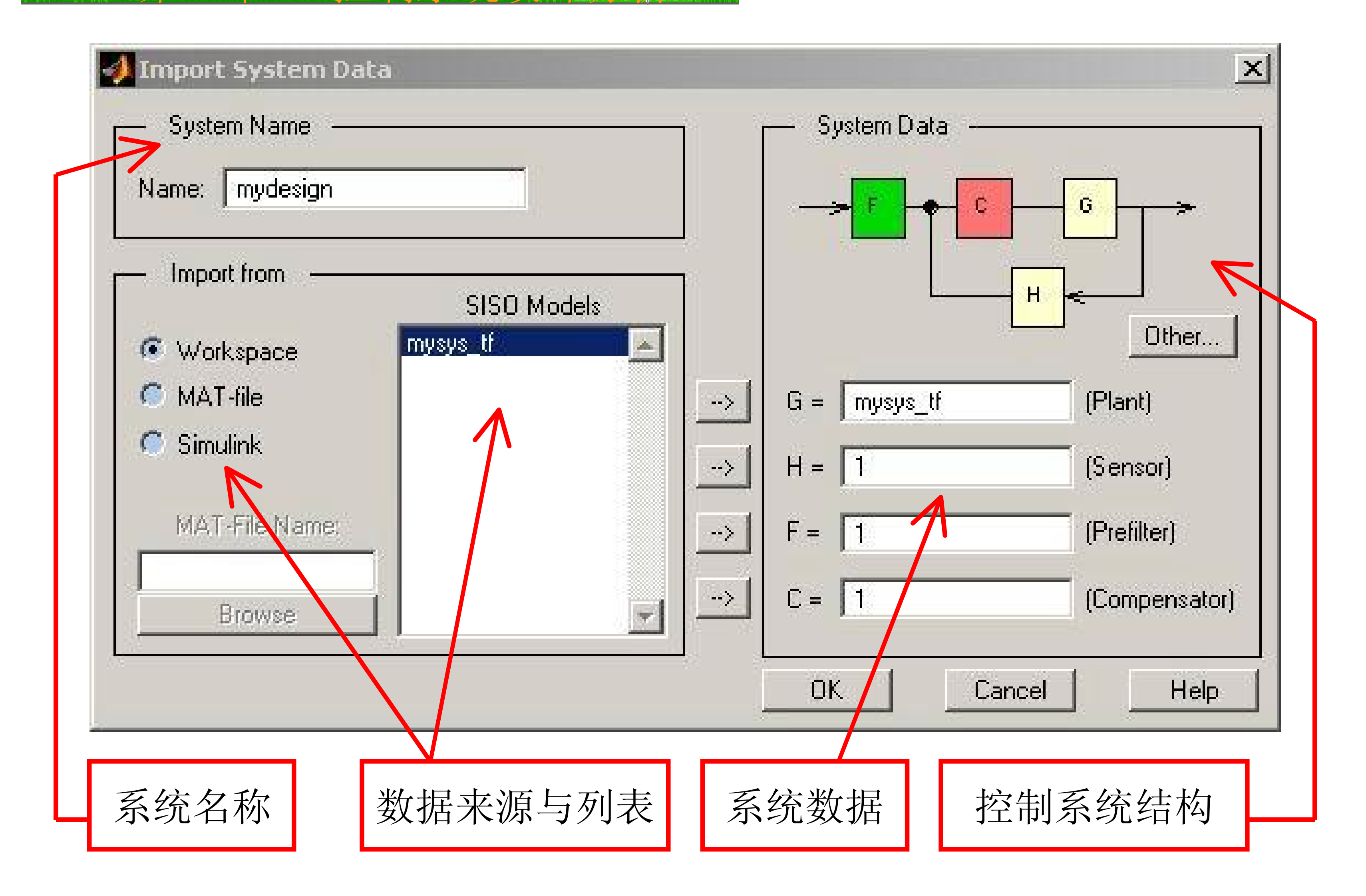


图10.19　系统数据输入对话框

使用此对话框可以完成线性系统的数据输入。注意，如果数据来源为Simulink系统模型框图，则必须对其进行线性化处理以获得系统的LTI对象描述。这是因为SISO线性系统中的所有对象（G执行结构、H传感器、F预滤波器、C补偿器）均为LTI对象。另外，用户可以单击控制系统结构右下方的Other按钮以改变控制系统结构。改变后的控制系统结构示意图如图10.20所示。

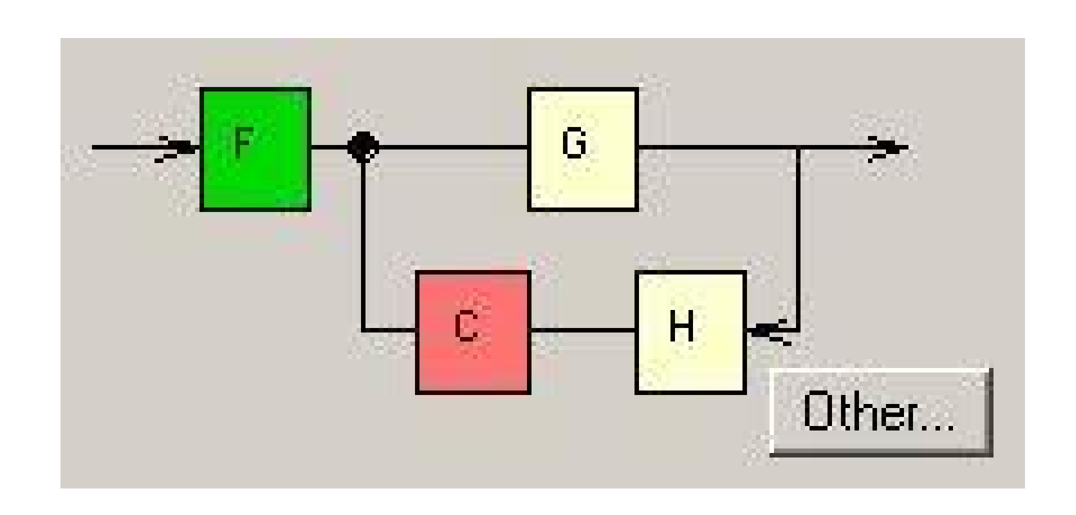


图10.20 控制系统结构改变示意图

使用SISO默认的控制系统结构，并设置控制系统的执行结构（即控制对象）数据G为mysys_tf，其它的参数H、F、C均使用默认的取值（常数1）。然后单击OK按钮，此时在SISO设计器中会自动绘制此负反馈线性系统的根轨迹图以及系统开环波特图，如图10.21所示。

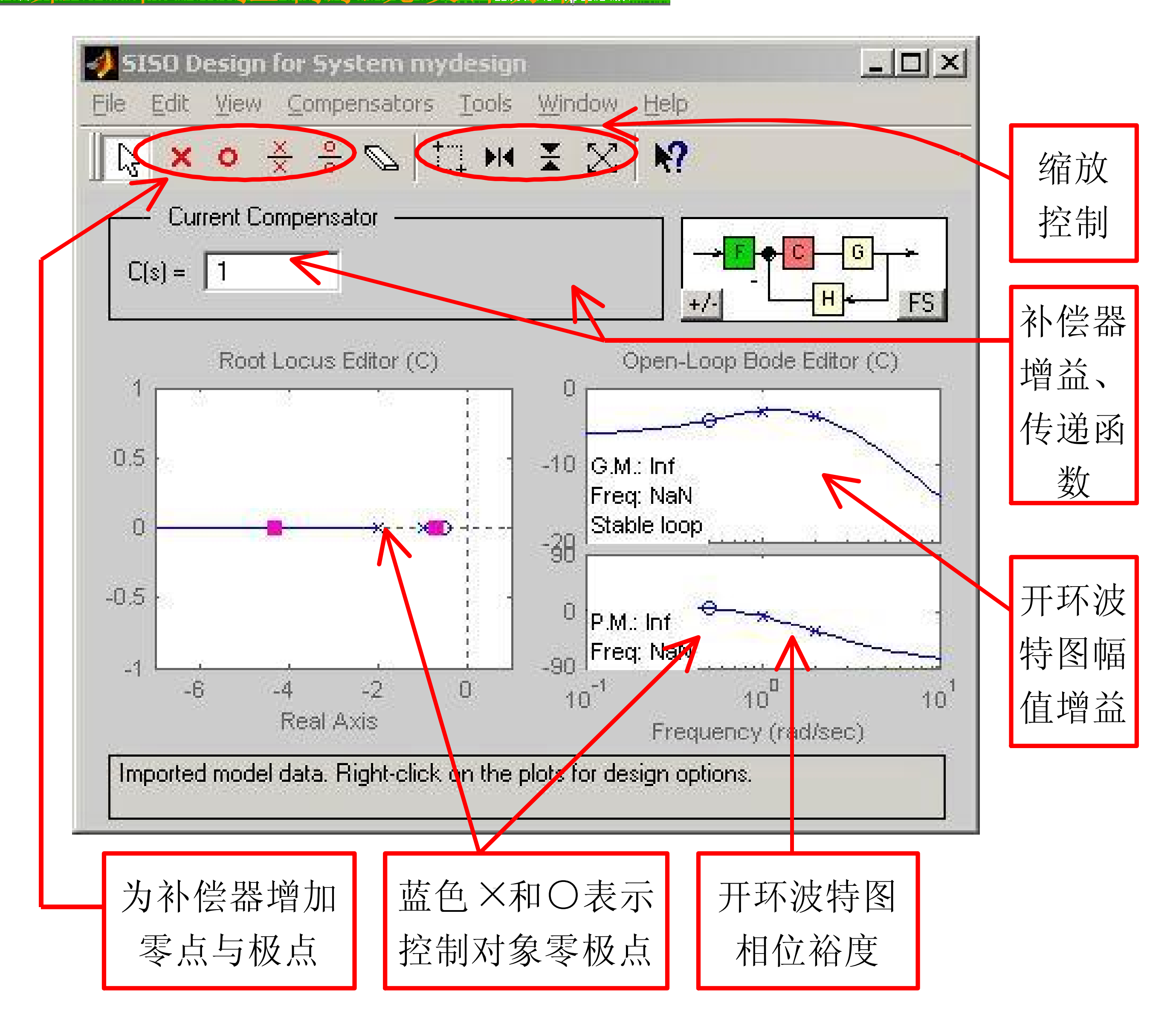


图10.21 系统数据输入后的SISO设计器界面

3. 设计与分析系统

在完成线性系统数据的输入之后，用户便可以使用诸如零极点配置、根轨迹分析以及系统波特图分析等传统的方法对线性系统进行设计。除了前面介绍的对系统零极点的各种操作（增加、删除以及改变分布）之外，SISO中对线性系统的设计提供了诸多的支持，如：单击补偿器增益及传递函数区域可以弹出补偿器设置对话框，使用此对话框可以设置补偿器C的增益、零点及极点等，如图10.22所示。

Edit Compensator C

Gain: 1　Format: Zero/Pole Location

Zeros
Delete　Real　Imaginary
Add Real Zero　Add Complex Zero

Poles
Delete　Real　Imaginary
-5.26
Add Real Pole　Add Complex Pole

OK　Cancel　Help　Apply

补偿器增益

补偿器极点

补偿器零点

图10.22 补偿器C的增益、零点及极点设置

限于篇幅，在此对SISO的操作不做过多的介绍，而主要介绍使用SISO设计器设计线性控制系统的思路与流程，以使用户能够对其有一个全面的认识，并且理解Simulink在实际系统设计分析中的应用。因此，在此系统设计中，我们仅仅为补偿器增加一个极点，如图10.23所示。

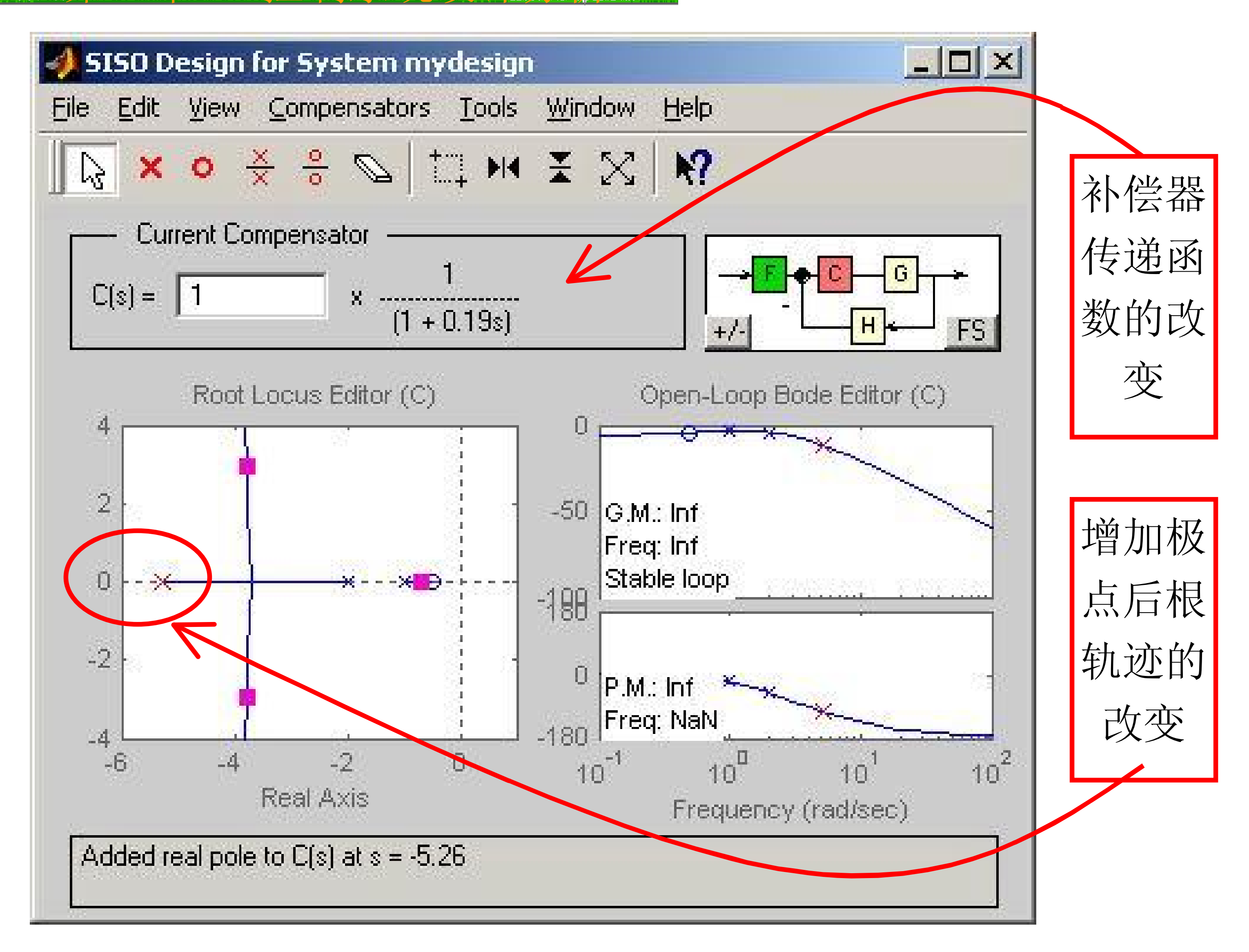


图10.23 为系统增加极点

在系统设计完成后，需要对其做进一步的分析：分析反馈系统的开环和闭环响应，以确保系统是否满足特定的设计要求。用户可以选择SISO设计器中Tools菜单下的Loop Responses绘制指定的开环响应（或闭环响应）曲线。此时将打开LTI浏览器，用户可在LTI浏览器中对系统的性能如过渡时间、峰值响应、上升时间等等进行分析，如图10.24所示。

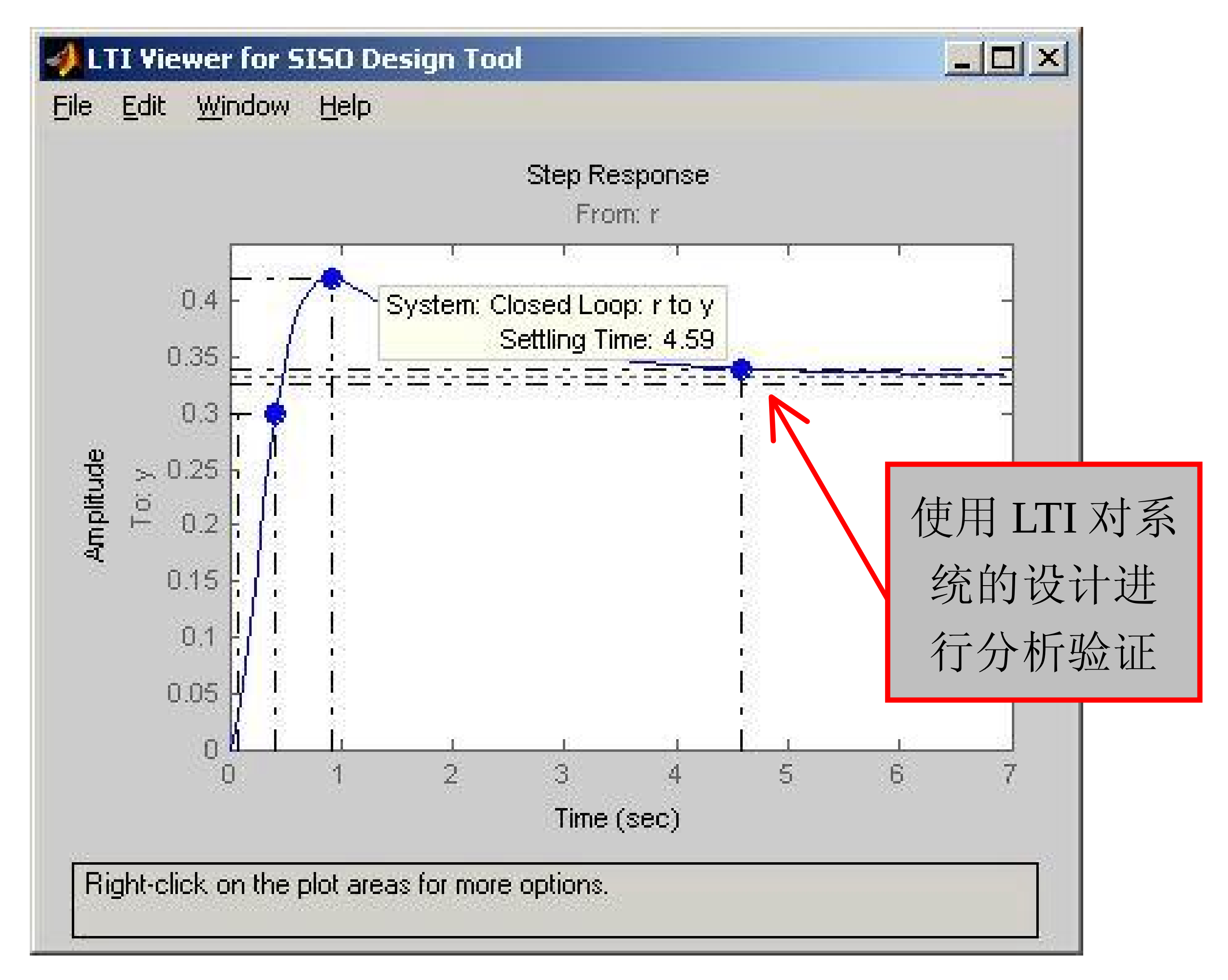


图10.24 使用LTI对系统的设计进行分析验证

如果用户需要设计线性离散控制系统，可以选择Tools菜单下的Continuous / Discrete Conversions选项，以对离散控制系统的采样时间、连续信号的离散化方法等进行设置，如图10.25所示。



图10.25　离散控制系统设计设置：采样时间与离散化方法

4. SISO设计器与Simulink的集成：系统验证

在使用SISO完成系统的设计之后，在系统实现之前必须对设计好的系统进行仿真分析，以确保系统设计的正确性。如果直接按照系统设计逐步建立系统的Simulink，将是一件很麻烦的工作；庆幸的是，SISO提供了与Simulink集成的方法，用户可以直接使用SISO设计器Tools菜单下的Draw Simulink Diagram直接由设计好的系统生成相应的Simulink系统框图。在生成Simulink系统模型之前，必须保存线性系统的执行结构、补偿器以及传感器等LTI对象至MATLAB工作空间中。图10.26所示为此系统相应的Simulink系统模型以及MATLAB工作空间变量列表。

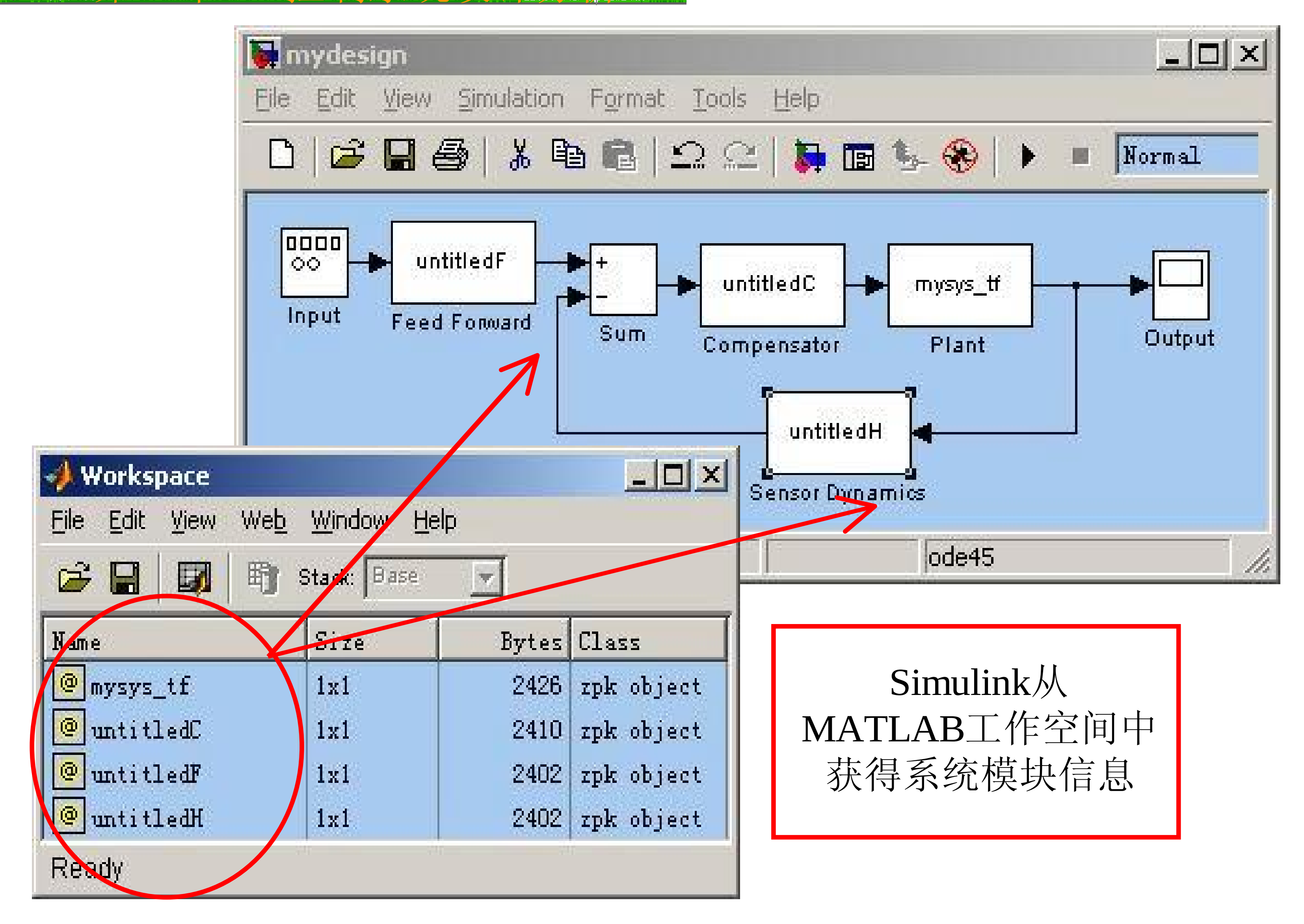


图10.26　由设计好的系统直接生成相应的Simulink模型

Output

0.5
0.4
0.3
0.2
0.1
0
0　2　4　6　8　10

Time offset:　0

与 LTI Viewer 浏览器中的结果一致

图10.27 Simulink系统模型的仿真结果



10.3 非线性控制系统设计简介

1. 非线性控制系统设计模块库

为了解决非线性系统线性化中存在的问题，用户可以使用非线性控制设计模块库NCD（Nonlinear Control Design Blockset）对非线性系统进行优化处理。用户可在指定的信号上连接一个NCD Outport模块，并确定对此信号的约束。NCD模块按照信号的约束优化非线性系统中控制器的参数，使系统能够满足约束的要求。

简单来说，NCD模块可以为系统设计人员提供如下的功能：

(1) 受时域约束和不确定性作用下模块的最佳参数。

(2) 对任何信号(驱动信号、传感器信号、命令跟踪误差等)施加约束。

(3) 在优化过程中可以随时停止，并查看中间结果。

2. 非线性系控制系统设计

在使用 Nonlinear Control Design Blockset之前，需要在Simulink中建立非线性控制系统（闭环形式）的系统模型。对于系统的各种模块（如控制器、补偿器等），无论是线性的还是非线性的，都应该在MATLAB的工作空间中定义这些模块的初始值信息。NCD Outport 模块必须连接到待控制的信号上。然后在这些模块上定义约束条件，这里的约束条件是针对连接点处的真实信号而言的，通常是一定幅值下的阶跃形式。在对非线性控制系统的设计完成之后，便可以使用NCD Outport模块对系统中各模块的参数进行优化，以使系统的性能能够满足给定的约束条件。

3. 约束窗口

在建立好非线性控制系统模型并运行系统仿真之后，双击NCD Outport以打开约束窗口。在约束窗口中显示三个图形：约束条件（由红色条显示）、系统在原来参数下的响应曲线（由白色曲线显示）以及中间响应（由绿色曲线显示）。如果系统的响应不满足约束（比如超出了黑色区域），优化继续进行。并且用户可以在任何时候停止优化并检查结果。至于对NCD Outport约束窗口的诸多操作与控制，这里不再介绍。最后，在关闭NCD Outport约束窗口时，将会出现一个对话框以提示用户保存优化后系统模型中的模块参数（保存到MATLAB工作空间中）。图10.28所示为某非线性控制系统的约束窗口。

初始参数下的响应曲线

约束下的优化结果

系统约束

图10.28 非线性控制系统的约束窗口



第11章 DSP Blockset



11.1 DSP处理单元：帧

11.1.1 基于帧的信号处理

大多数实时的数字信号处理系统都采用基于帧的处理方式，以提高系统性能，这里每帧包含相邻的多个或者一组信号采样。采用基于帧的处理方式更适合多数的数字信号处理算法，另外也可降低系统对数据采集硬件的要求。缺省情况下，Simulink所有信号都是基于采样的。

表11.1 基于采样的信号和基于帧的信号

基于采样的信号	基于帧的信号
每个时间步处理一个采样点	每个时间步处理含有 N 个采样点的一帧
仿真步长 = 采样周期 = T_s	仿真步长 = 帧周期 = $N*T_s$
采样频率 $F_s = 1/T_s$	帧频率 = F_s / N
采样步长可变	帧的大小可变，可以是工作区中的一个变量

之所以采用基于帧的处理主要是考虑到数字信号处理本身的要求和数据通讯的开销。显然，基于帧的信号处理应当比基于采样的处理要复杂得多，但是Simulink利用MATLAB的矩阵功能极大地提高了处理的效率。通过基于帧的处理，减少了块与块之间的通讯，从而比使用基于采样的信号进行仿真快得多。总之，利用基于帧的信号提高了仿真速度。而且，由于同样的原因，大多数DSP系统也采用基于帧的处理。除此之外，基于帧的处理提供了在仿真中进行频域分析的能力。

Simulink的所有模块都支持基于帧的处理，使得用户可以方便地采用基于帧的信号进行算法仿真以及结合RTW产生实时代码。

图11.1说明了从连续信号经过AD采样得到采样信号，然后将采样信号组织成帧，送往Simulink处理的过程。

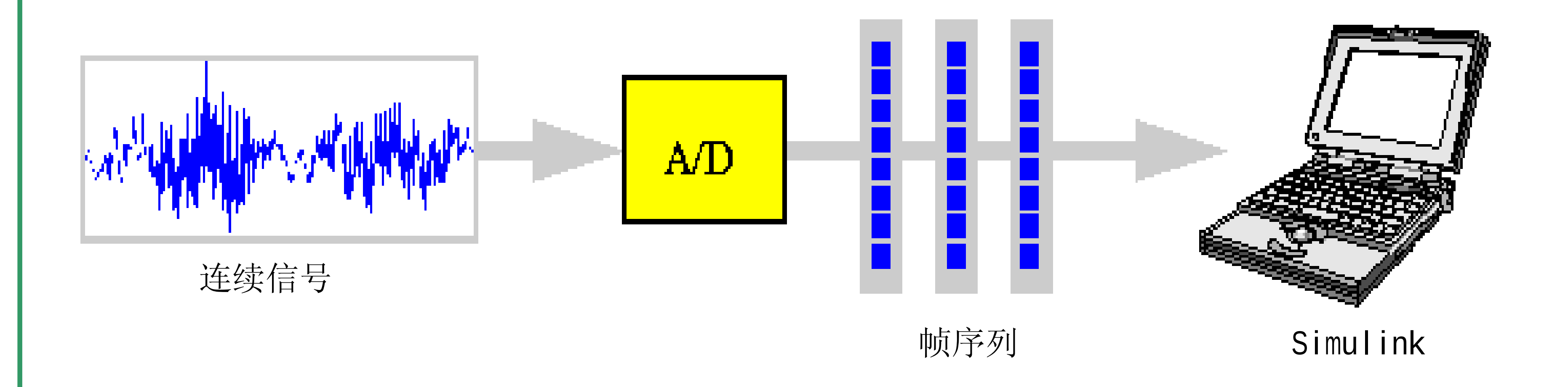


图11.1 基于帧的信号处理

1. 缓冲和解缓冲

在Simulink中采样信号和帧信号之间的转换是通过缓冲模块（Buffer）来实现的。Buffer 块有两种用途：一是接受采样输入并产生一定帧大小的帧输入；二是接受帧输入，修改帧的大小，这种情况下必须使用缓冲模块。这两种情况下都涉及到帧之间的重叠和帧的初始值的设置问题。当通过采样产生帧时，缓冲使用输入标量生成一个列向量，如图11.2所示。如果需要从一个帧信号产生一个采样信号，则应使用Unbuffer模块。

Source库中的许多信号源模块同样提供基于帧的输出，当然使用这些模块作为输入信号时，就无需使用Buffer块，只需设置块的帧长参数就可以了。

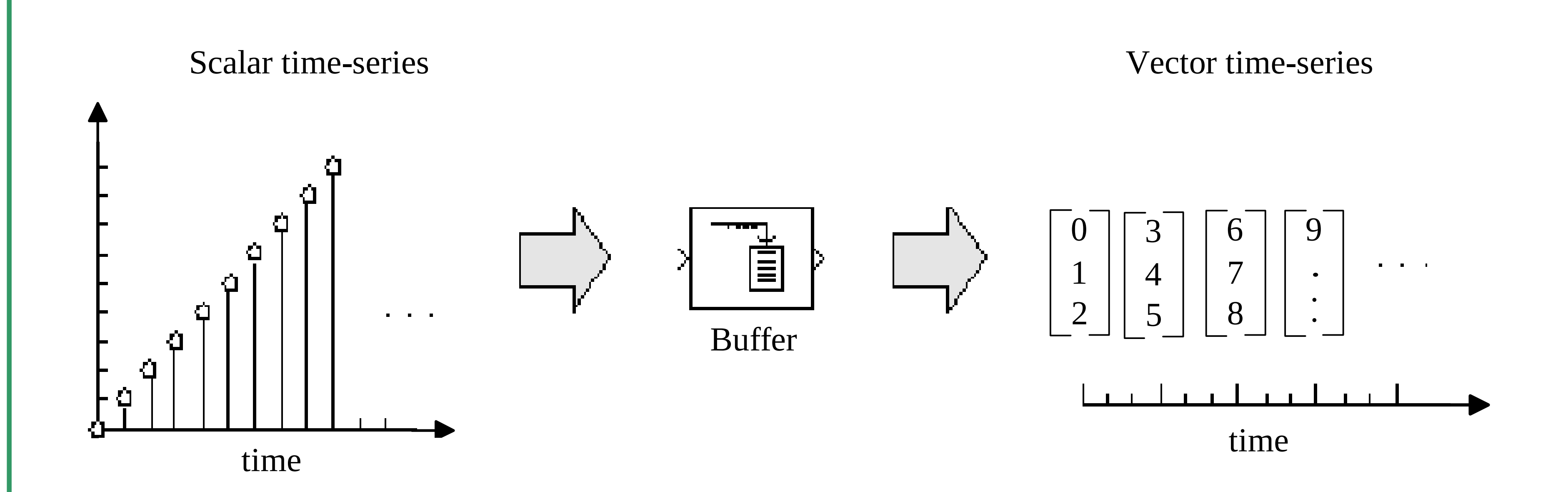


图11.2　缓冲模块

2. 帧的表示

通常，一帧是通过一个矩阵表示的。在帧矩阵中，每个通道的信号对应矩阵中的一列，每个采样对应其中的一行（如图11.3所示）。在基于帧的处理中，各个模块沿着输入的每一列（通道）进行运算。图11.3中有四个信号通道，每帧有两个采样，帧和帧之间没有重叠。通常每帧的采样数是2的幂次，以满足FFT变换的需要。

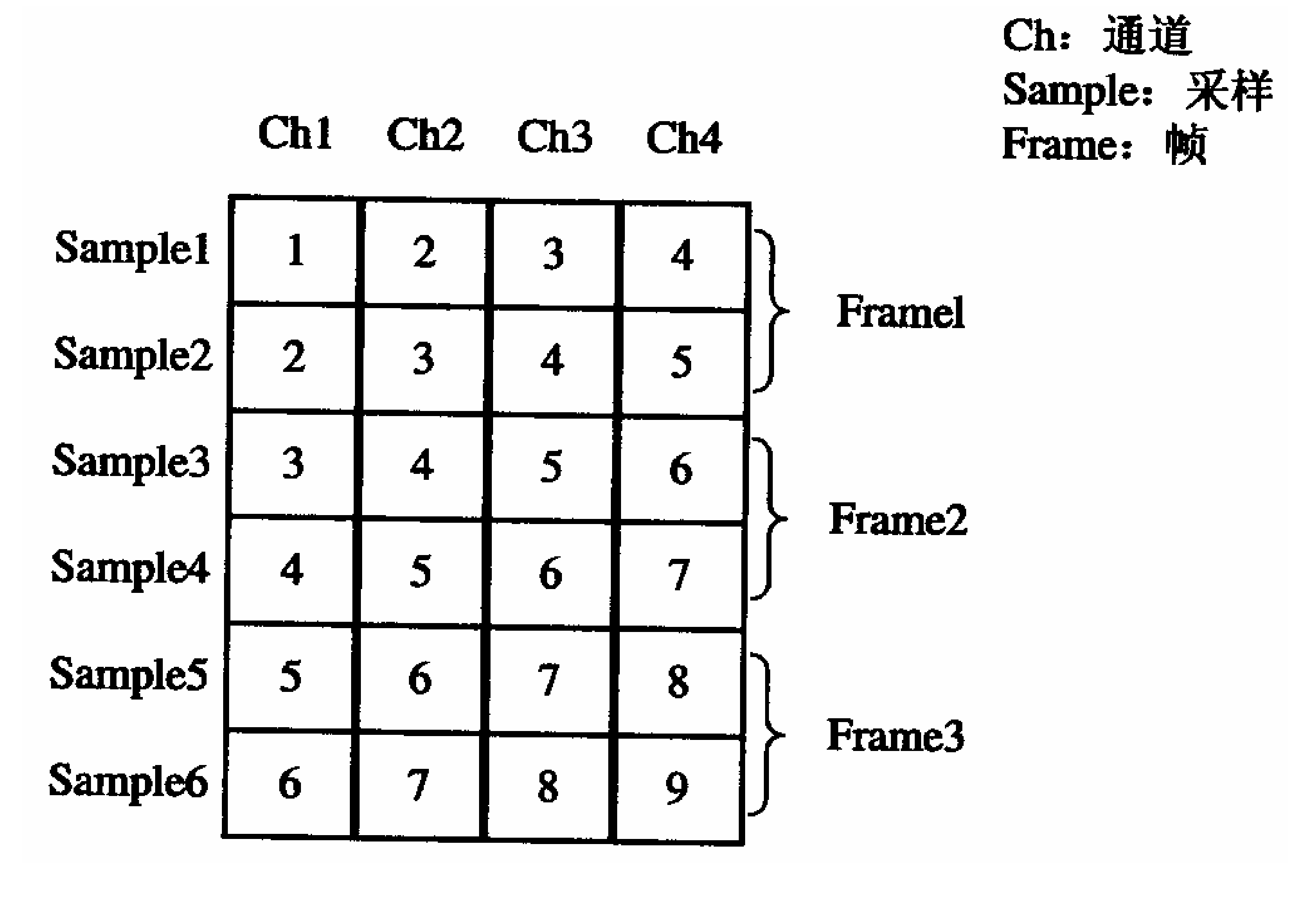

Ch：通道
Sample：采样
Frame：帧
Ch1
Ch2
Ch3
Ch4
Sample1
Sample2
Sample3
Sample4
Sample5
Sample6
1 2 3 4
2 3 4 5
3 4 5 6
4 5 6 7
5 6 7 8
6 7 8 9
Frame1
Frame2
Frame3

图11.3 帧矩阵

3. 生成基于帧的信号

主要有三种方法用来生成基于帧的多通道信号。

(1) DSP模块库中信号源库DSP Sources中的块提供了信号源块，用于生成基于帧的信号。

(2) 所有的信号都可以通过缓冲块成为帧。

(3) 将从若干个基于帧的信号源来的信号通过矩阵拼接成一个帧矩阵，形成一个多路信号。

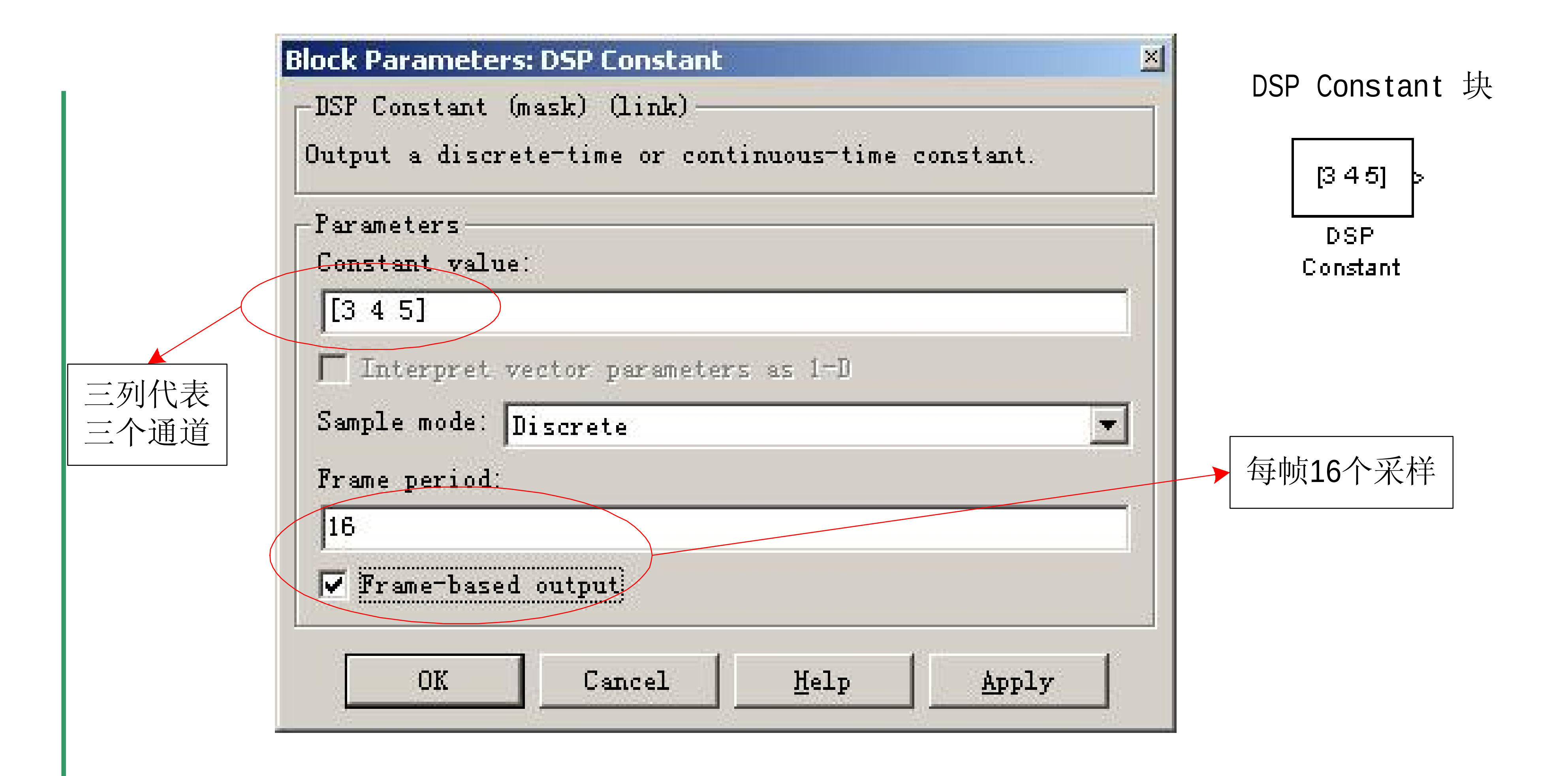


图11.4 DSP Constant 模块设置

4. 观察基于帧的信号

用户可以使用DSP blockset提供的专门的显示模块来观察基于帧的信号。这些模块中最常用的是Matrix Viewer（矩阵浏览器）和Vector Scope（向量示波器）。Matrix Viewer将输入矩阵的行和列作为坐标轴，使用不同颜色表示矩阵元素的值，还可以根据需要自己建立一个颜色表。Vector Scope 显示输入的每一列（通道），按照指定帧的数目每次显示整个数据。Vector Scope 可以显示时域或频域信号。图11.5是基于帧的三个正弦信号（三个通道）分别用Matrix Viewer和 Vector Scope显示的结果。此外还有内置FFT变换的Spectrum Scope用来直接显示时域信号的频谱。

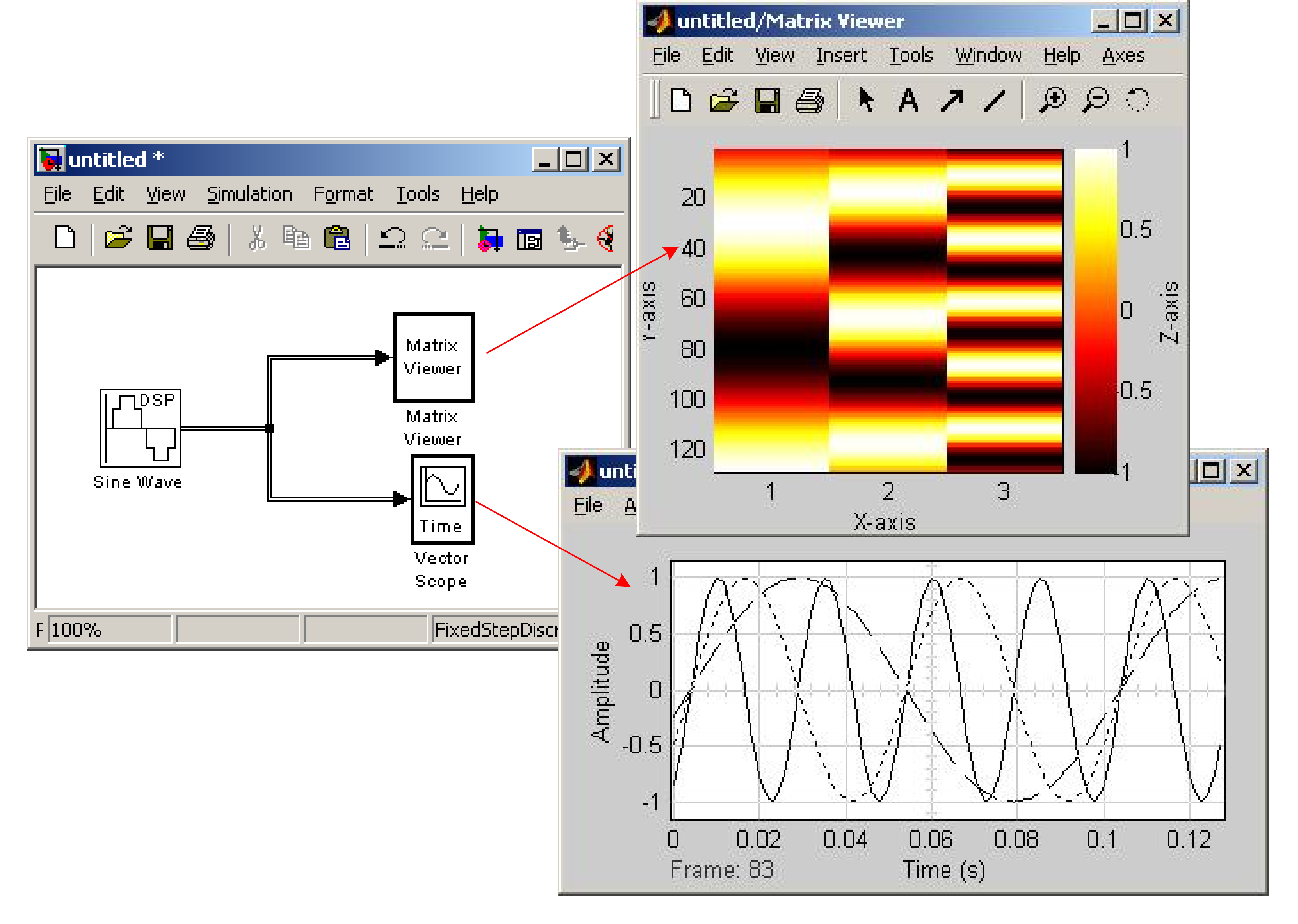


图11.5 Matrix Viewer & Vector Scope

5. 使用基于帧的信号

当一个信号线表示基于帧的信号时，Simulink用双线来绘制。基于帧的信号处理可以使用Simulink中对输入的每个元素进行处理的块，但是不能使用Simulink中对向量处理的模块（例如Unit Delay和Mux）。实际上这些模块中许多模块在DSP Blockset中都有一个与之对应，专门用来做基于帧的信号处理的版本。例如，在DSP Blockset中等价于Unit Delay的模块是Integer Delay模块，与Mux等价的模块是 Matrix Concatenation模块。图11.6所示的框图是对随机信号延迟30个步长后进行卷积处理。下面给出一个具有回响功能的声学例子，读者不妨一试。

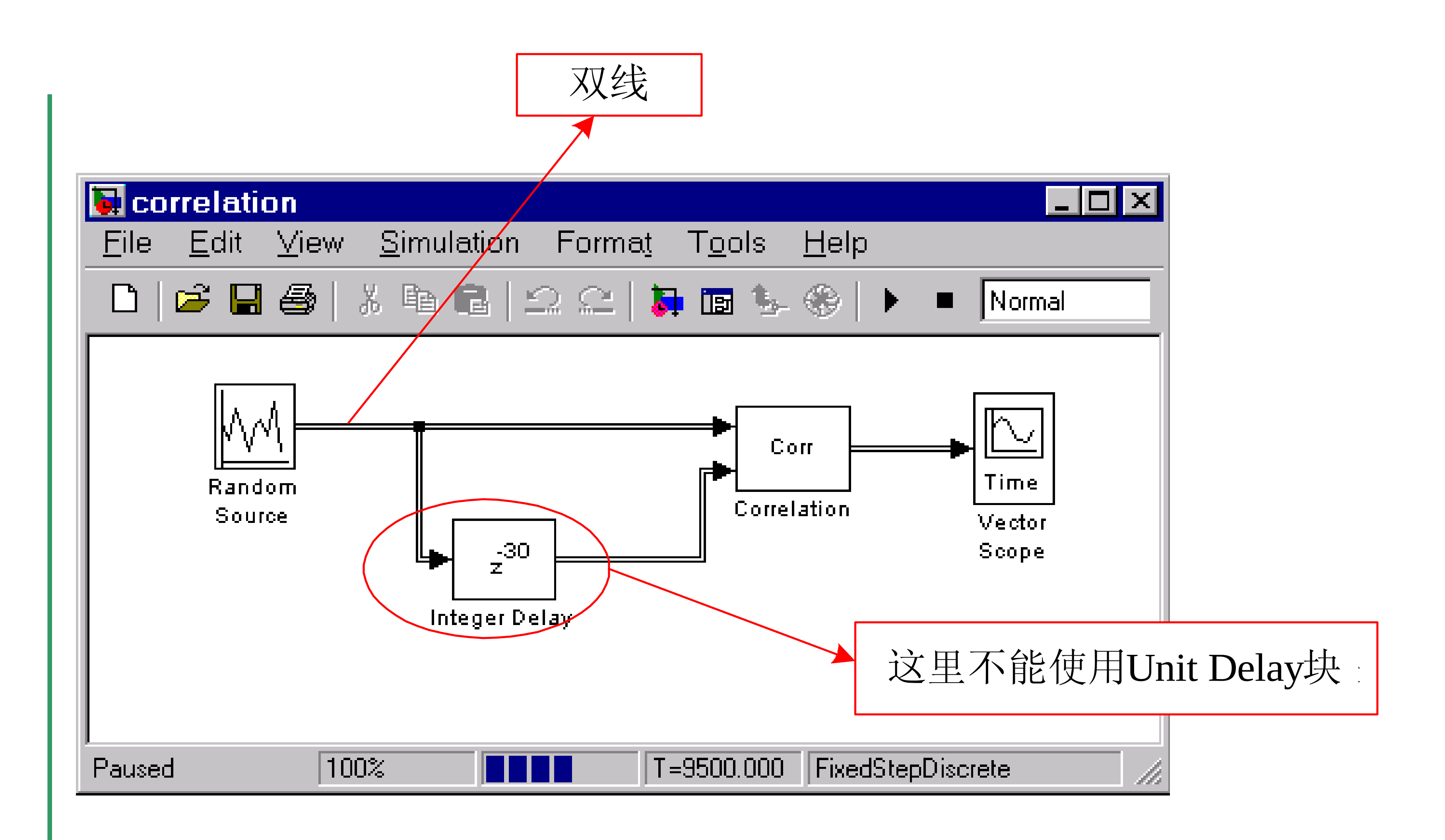


图11.6 基于帧的信号处理

【例11.1】　试建立一个圆形剧场的声学模型，假设有70%的信号被反射回来。

解：使用\DSP Sources\From Wave File模块加载一个声音文件(*.wav)，采样频率为8000 Hz。圆形剧场的回声效果导致70%的信号在2 s之后反射回来，这里用一个增益为0.7的Gain模块表示。其中使用一个Integer Delay模块产生2×8000的采样延迟。最后使用一个\DSP Sinks\To Wave Device 模块听一下效果。回声系统模型如图11.7所示。注意，To Wave Device 模块只能在PC平台上使用。

acoustics

File Edit View Simulation Format Tools Help

Normal

From Wave File
audio.wav
(8000Hz/1Ch/16b)

From Wave
File

Gain

0.7

z^{-1600}

Integer Delay1

To Wave
Device

Ready 100% FixedStepDiscrete

图11.7 回声系统模型

11.1.2 设置Simulink进行DSP仿真

对于一般系统而言，Simulink的默认设置认为信号是连续的，而且使用连续变步长求解器对系统进行求解。如果系统中包含连续信号和离散信号，应当使用此配置。但是对于纯离散的数字信号处理系统的设计、仿真与分析而言，需要对Simulink重新配置，使其能够适用于数字信号处理。

用户可以使用M文件dspstartup来配置Simulink，使之适用于数字信号处理。设置的内容包括：使用固定步长求解器、在采样之间信号没有定义（避免两个不同采样率信号之间的操作）、结束时间设为无穷大、仿真时间和数据不保存到工作区以节省内存等等。此外，用户还可以根据需要修改dspstartup.m文件以加入定制的设置。图11.8为设置好的DSP仿真参数页面。

如果经常需要进行DSP仿真，用户可以在startup.m文件中加入dspstartup命令，MATLAB在启动后自动运行startup，这样就无需每次仿真都运行dspstartup命令了。

Simulation Parameters: untitled

Solver | Workspace I/O | Diagnostics | Advanced | Real-Time Workshop

Simulation time

Start time: 0.0 Stop time: inf

Solver options

Type: Fixed-step discrete (no continuous states)

Fixed step size: auto Mode: SingleTasking

Output options

Refine output

OK

数字信号处理系统是离散时间系统，不含连续状态

要使用RTW产生实时代码，此项必须设置为Fixed-step

Simulation Parameters: untitled

Solver | Workspace I/O | Diagnostics | Advanced | Real-Time Workshop

Load from workspace

Input: [t, u]

Initial state: xInitial

Save to workspace

Time: tout

States: xout

Output: yout

Final state: xFinal

Save options

Limit data points to last: 1000

Decimation: 1

Format: Array

OK Cancel Help Apply

不存储时间状态等变量以减少对内存的占用

图11.8 设置Simulink进行DSP仿真



11.2　DSP Blockset模块库介绍

DSP Blockset库提供了极为丰富的DSP模块资源，它们封装了几乎所有基本的数字信号处理操作和算法，其中的许多模块在信号处理工具箱中都有对应的函数。用户可以利用这些模块方便地完成自己的数字信号处理系统仿真和分析。图11.9列出了展开后的DSP Blockset模块库。这一节将分别介绍各个子库并给出一些简单的例子。

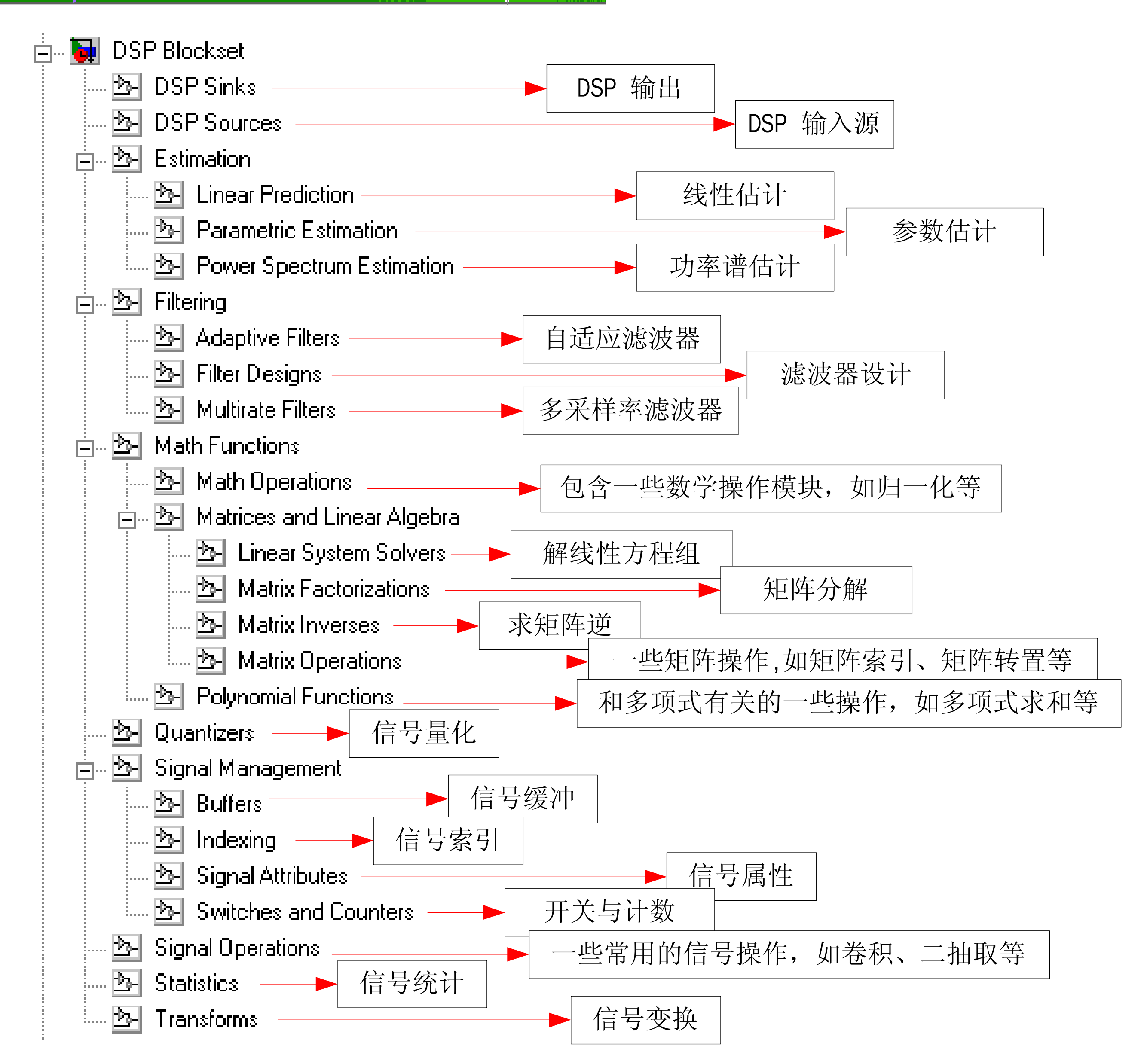


图11.9 DSP Blockset模块库

11.2.1 信号的操作和管理

一般的信号操作如加窗和补零可以通过Signal Operations模块库完成。这个库中还包含Variable Interge Delay模块，这里延迟的大小是通过第二个输入信号指定的。表11.2列出了各个Signal Operations 库中各个模块及其功能描述。

表11.2 Signal Operations 库

模块名称	模块功能描述
Convolution	计算两个输入的卷积
Downsample	抽取操作
Upsamole	插值操作
Interger Delay	延迟操作
Pad	补值操作
Zero Pad	补零操作
Repeat	重复操作，重复输入采样*N*次
Sample and Hold	当收到一个触发信号后，对输入信号采样并保持直到收到下一个触发信号
Unwrap	展开信号的相位
Variable Fractional Delay	按照一个变量的值延迟每个通道的信号，该变量可以是分数
Variable Integer Delay	按照一个变量的值延迟每个通道的信号，该变量是整数
Window Function	加窗操作，可以选择不同的窗类型

信号可以通过在Signal Management下的四个库进行一些管理操作，它包括缓冲、索引、信号属性、切换与计数四个部分，如表11.3所示。

表11.3 Signal Management库

子库名称	模块名称	模块功能描述
Buffers	Buffer	缓冲
	Unbuffer	解缓冲
	Delay line	重新缓冲信号，每次更新一个采样
	Queue	FIFO（先入先出）寄存器
	Stack	实现一个栈，或者先入后出寄存器
	Triggered Dealy line	带有使能端的Delay line
Indexing	Flip	按行或者按列倒置矩阵或者向量
	Selector	从矩阵或者向量中选择元素
	Multiport Selector	从矩阵或者向量中选择多组元素
	Variable Selector	按照输入变量选择元素
	Submatrix	从一个矩阵中选择一个子矩阵

Signal Attributes	Check Signal Attributes	检查信号的属性是否符合设置
	Contiguous Copy	将非连续存储的信号转换为连续存储的信号
	Convert 1-D to 2-D	将一维信号转换为二维信号
	Convert 2-D to 1-D	将二维信号转换为一维信号
	Frame Status Conversion	设置输出帧的状态
	Inherit Complexity	根据参考信号改变输入信号的表示形式（复数或实数形式）
Switch and Counters	Counter	脉冲计数器
	Edger Detector	边缘检测器，当信号变为0，或者从0变为其它值时，输出1
	Event-Count Comparator	统计非零输入的个数，当大于设置的数目时，输出变为1
	Multiphase Clock	产生相位依次移动的时钟信号阵列
	N-Sample Enable	经过*N*个采样后输出从0变为1
	N-Sample Switch	经过*N*个采样后输出从下端口输入变为上端口输入

信号的速率转换可以通过Signal Operations中的Upsample和Downsample模块来进行。Upsample通过在新的数据点上补零来实现，Downsample通过间隔去除部分采样点来降低采样速率。如图11.10所示，用一个probe模块来探测信号的采样速率，原始采样信号为1000 Hz，经过二抽取后变为500 Hz，经过插值后变为2000 Hz。

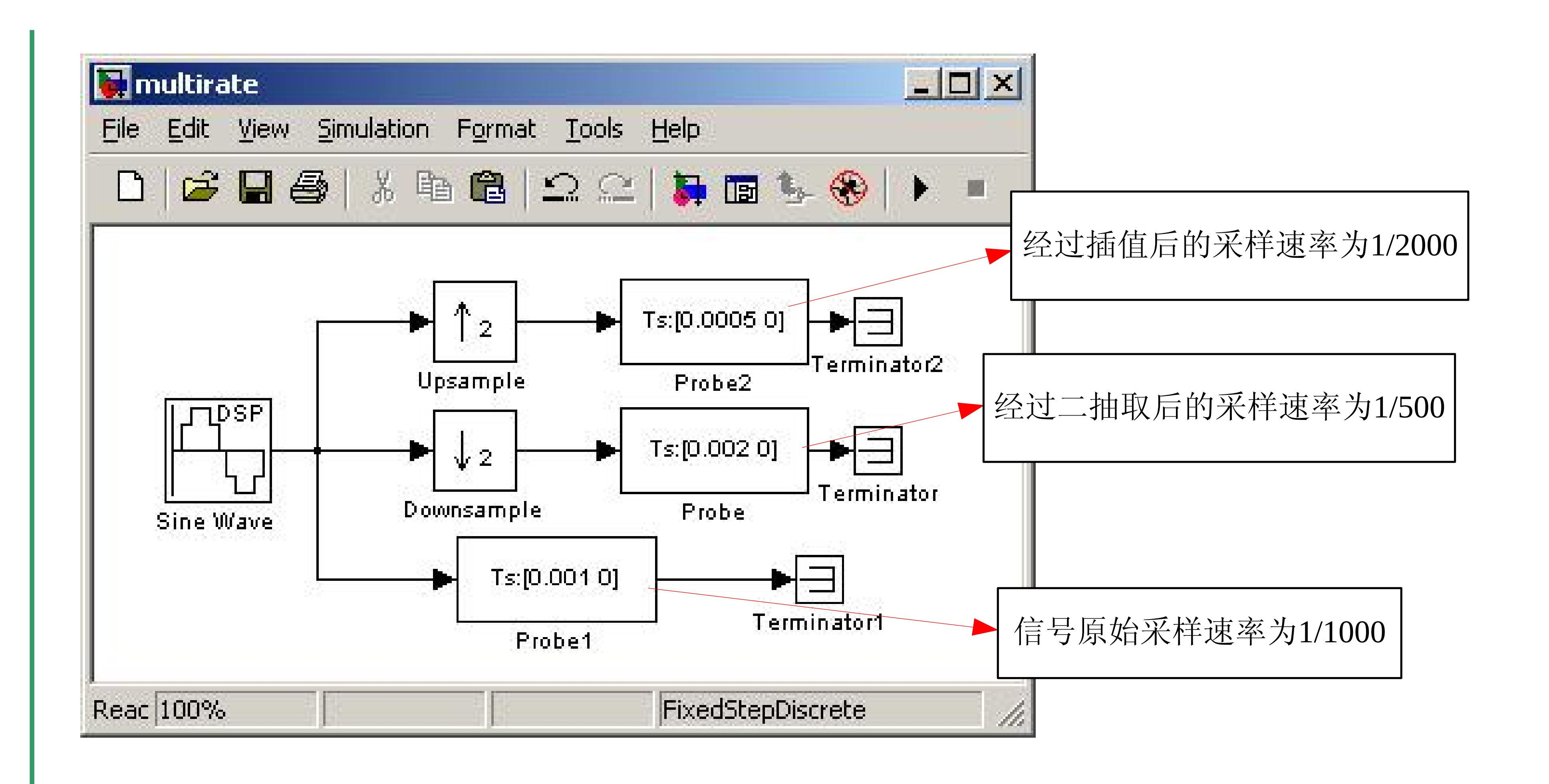


图11.10 改变信号的速率

11.2.2 信号变换

使用基于帧的信号的一个好处就是能够进行各种变换处理，这意味着可以得到关于信号特性的更多信息。Simulink DSP Blockset 提供了时域至频域，频域至时域（逆变换）和时域至时域的转换模块。需要注意的是，这些变换模块只能用于基于帧的输入的场合。表11.4列出了Transform 库中的模块及其功能描述。

表11.4 Transform 库

模块名称	模块功能描述
DCT	计算每个通道的离散余弦变换
FFT	计算每个通道的快速傅立叶变换
IDCT	计算每个通道的离散余弦反变换
IFFT	计算每个通道的离散傅立叶反变换
Analytic Signal	计算每个通道的分析信号
Complex Cepstrum	计算每个通道的复倒谱
Real Cepstrum	计算每个通道的实倒谱
Magnitude FFT	计算每个通道输入的幅度谱或者幅平方谱

【例11.2】 计算信号的频率响应。

解：有两种简单的方法可以用来计算信号的频率响应。一种是对输入信号做FFT变换，然后在一个Vector Scope中观察变换的结果，这时Vector Scope的输入域应设置为Frequency。在Vector Scope模块前需要插入一个 Complex to Magnitude 模块，因为向量示波器期待的是一个实型输入。另外一个方法是将信号直接接到一个FFT示波器上，这个示波器先进行FFT变换和求平方，然后再显示。计算信号的频率响应的框图如图11.11所示。输入信号是两个频率分别为50 Hz和100 Hz的正弦信号：

$$u(t) = [\sin(100\pi\, t) \quad \sin(200\pi\, t)]$$

各个模块的参数设置如下：

(1) Sine wave 模块：Frequency(Hz)设置为[50 100]。

(2) Gain 模块：Gain设置为0.5。

(3) Complex to Magnitude 模块：Output设置为 Magnitude。

(4) Vector Scope：Input domain设置为Frequency。

其它使用缺省设置。

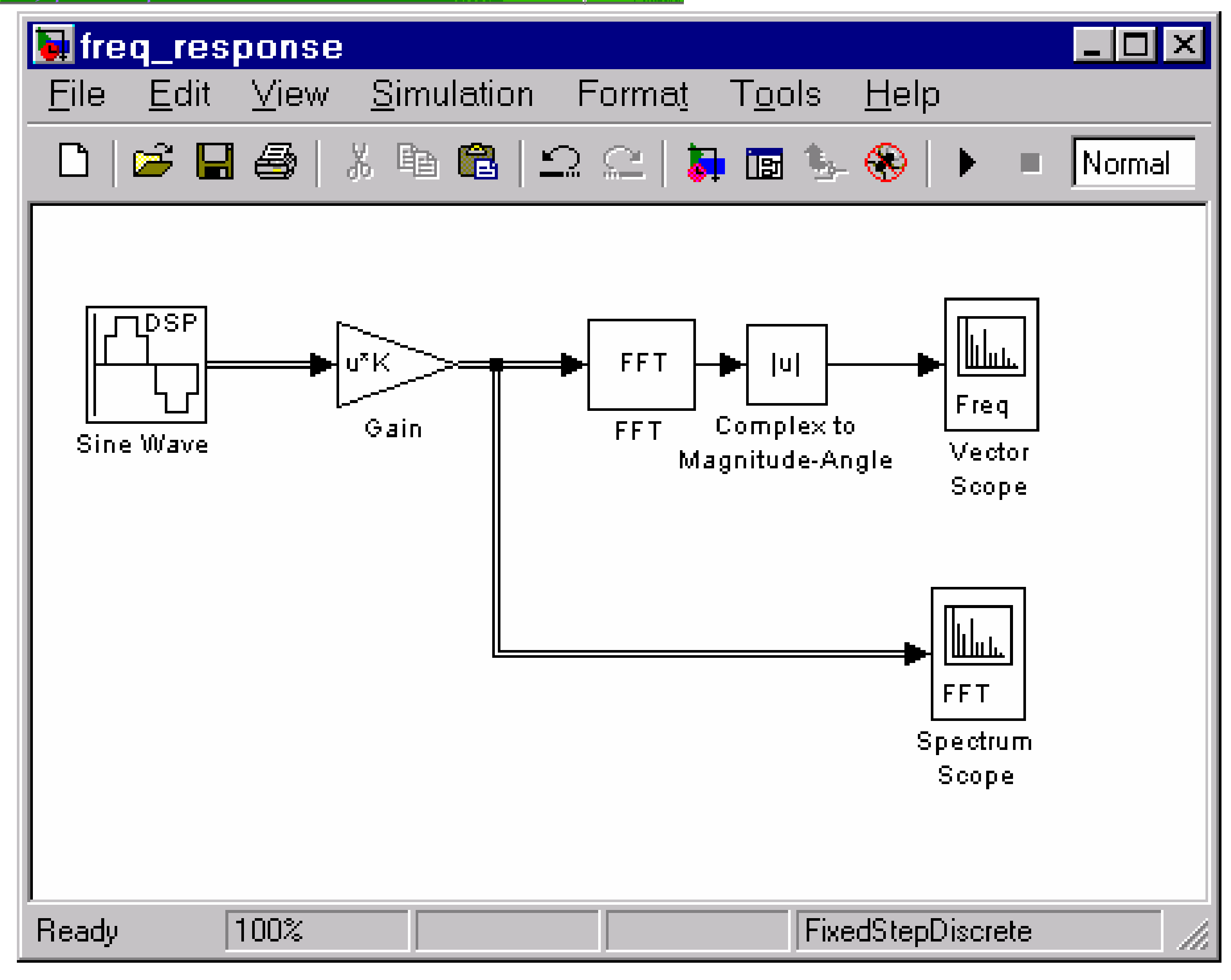


图11.11 计算信号的频率响应

11.2.3 滤波器设计与频率分析

有关滤波器设计和分析的内容非常丰富，在此仅对\Simulink DSP Blockset\filtering库中的模块进行简单的介绍，然后利用Simulink所提供的工具设计一个最简单的低通滤波器。

1. 设计一个滤波器

DSP Blockset库中有丰富的滤波器设计模块，可以设计数字FIR和IIR滤波器、模拟IIR滤波器。对于滤波器设计，应给出滤波器阶数和截止频率。实际上滤波器设计模块是通过信号处理工具箱(Signal Processing Toolbox)进行滤波器设计的，然后返回一系列滤波器系数。

Simulink 4.0为用户提供了两个非常好用的模块用于模拟滤波器和数字滤波器的设计。Analog Filter Design模块用于模拟滤波器的设计，只需在模块对话框中选择要设计的滤波器类型和方法及其它阶数等信息就可以了。Digital Filter Design模块用于数字滤波器的设计，双击该模块可以看见如图11.12所示的图形化的设计界面，通过它用户可以方便地进行各种常用数字滤波器的设计和分析，设计完后可以直接作为滤波器的实现模块在仿真中使用

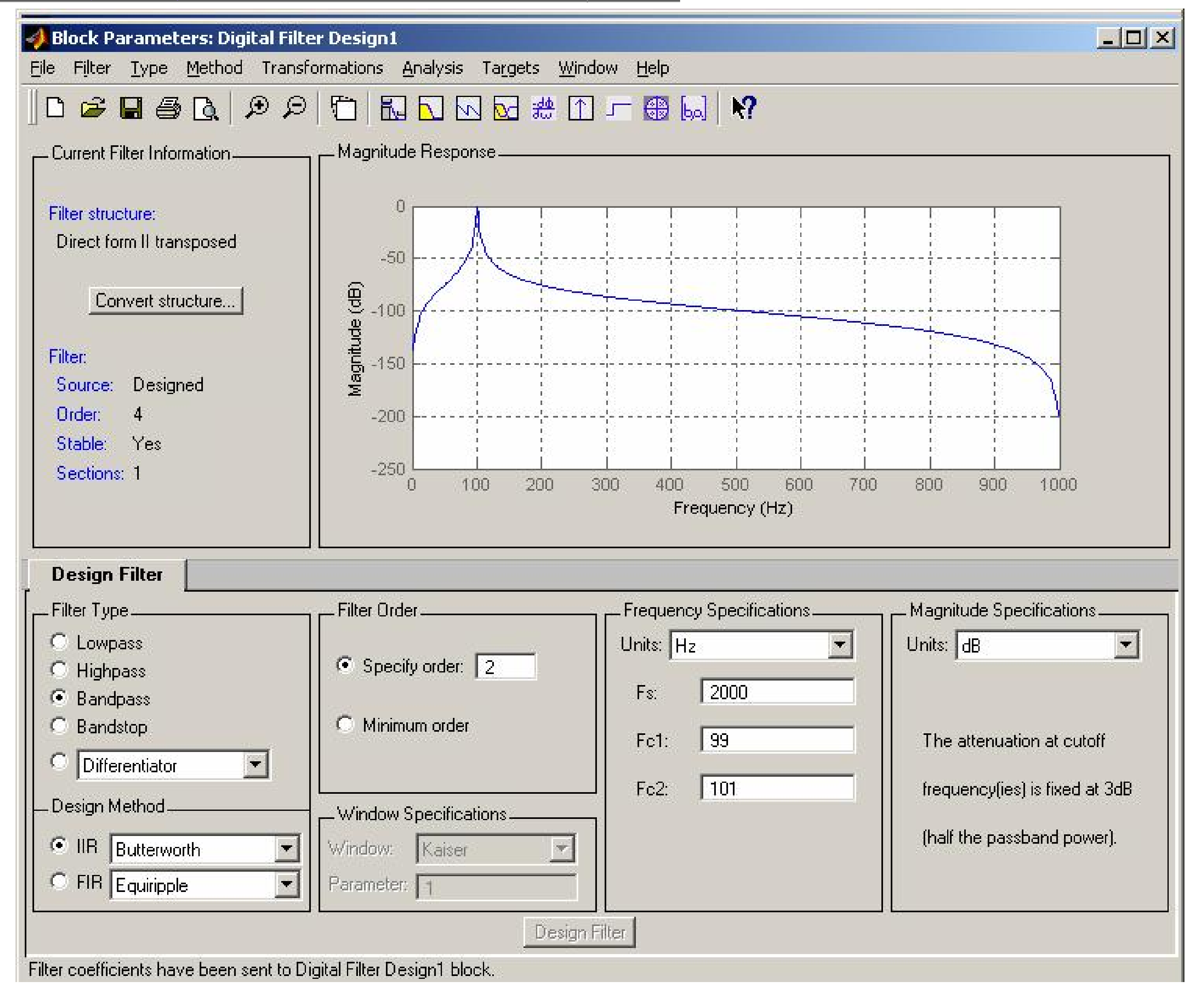


图11.12 滤波器设计工具：FDATool

【例11.3】 设计一个滤波器滤除正弦信号中的噪声，还原正弦信号。正弦信号为sin100*t*，噪声信号是均值为0、方差为1的高斯白噪声。信号采样频率为2000 Hz。

解： (1) 首先用Digital Filter Design模块设计Butterworth带通滤波器。各个选项参数的设置如图11.12所示。然后单击Design Filter按钮，完成滤波器的设计。

(2) 按照图11.13选择和连接好其余各个模块。各个模块的设置如下：

Samples per fame设置为256。

② Sine Wave 模块：Frequency设置为100，Sample Time设置为1/2000，Samples per fame设置为256。

③ Vector Scope模块：为缺省设置。

运行仿真，产生的结果如图11.13所示。从图中可以看出，正弦信号被清楚地还原了出来。

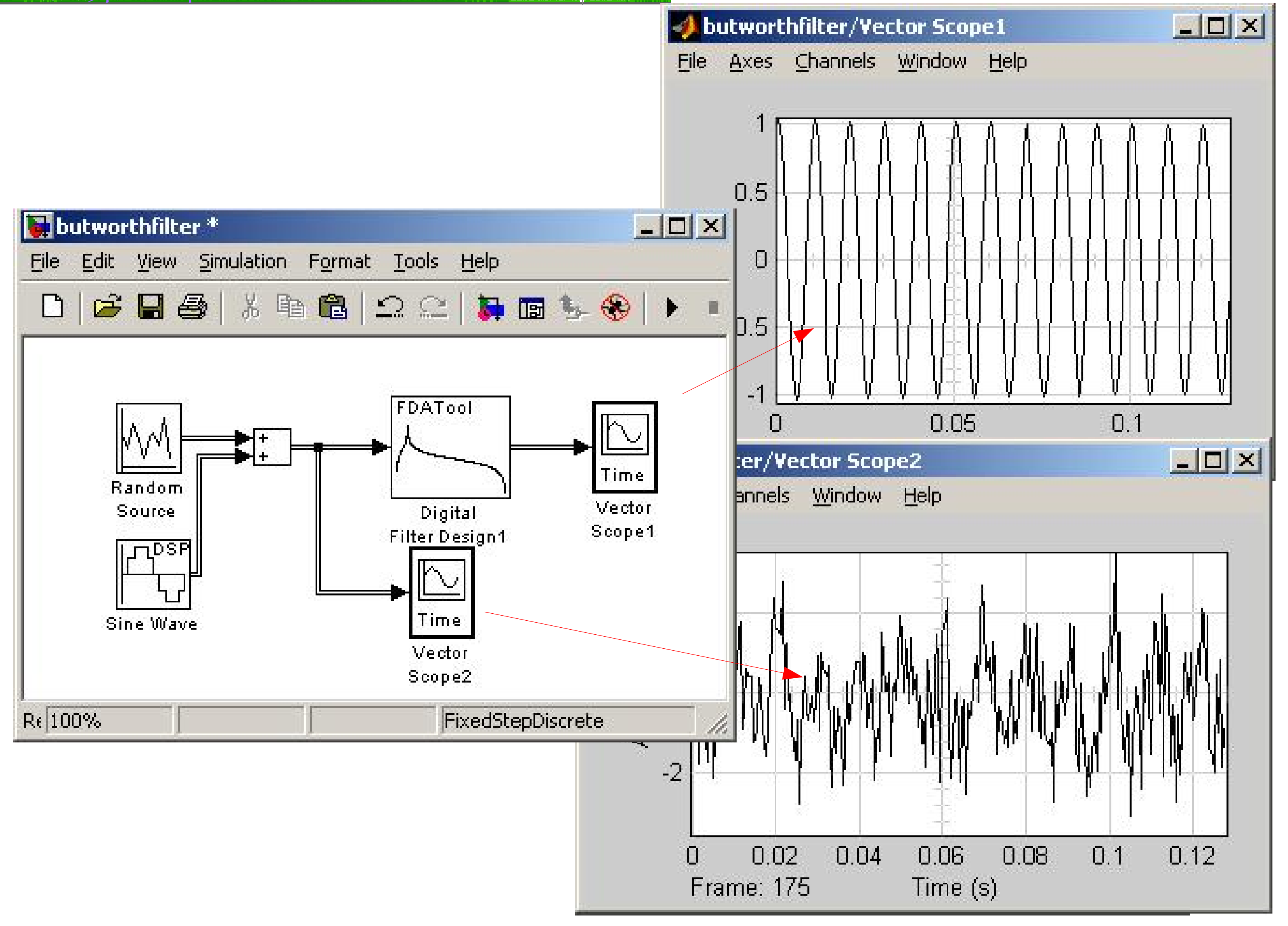


图11.13 带通滤波器结果输出

2. 实现一个滤波器

通过一个给定的传递函数表达的滤波器，可以只采用延迟和增益模块来实现。这个实现不是唯一的，而且对于FIR和IIR滤波器有不同的实现方法。滤波器系数可以直接在滤波器实现块中输入，或者使用Signal Processing Toolbox中的函数butter、fir1等产生。注意，设计这些滤波器时，截止频率往往是通过归一化频率来表示的。

11.2.4 功率谱估计

非参数估计方法和参数估计方法。非参数估计方法直接使用信号进行功率谱估计；参数估计方法试图建立一个等效系统并估计出系数。\DSP Blockset\Estimation\Parametric Estimation 库给出了几种AR模型参数估计方法；\DSP Blockset\Estimation\Power Spectrum Estimation库则给出了利用Parametric Estimation 库中的参数估计模块进行功率谱估计的模块，以及两种利用非参数估计方法估计信号功率谱的模块。

表11.5 Statistics 库

模块名称	模块功能描述
Autocorrelation LPC	自相关线性预测
Yule-Walker AR Estimator	利用Yule-Walker方法估计AR模型参数，它使用Levinson-Durbin递推算法解Yule-Walker方程
Burg AR Estimator	利用Burg方法估计AR模型参数
Covariance AR Estimator	利用协方差方法估计AR模型参数
Modified Covariance AR Estimator	利用改进的协方差法估计AR模型参数

Burg Method	利用Burg方法估计信号功率谱，它用到了Burg AR Estimator块
Covariance Method	利用协方差方法估计AR模型参数，它用到了Covariance AR Estimator块
Modified Covariance Method	利用改进的协方差方法估计AR模型参数，它用到了Modified Covariance AR Estimator块
Yule-Walker Method	利用Yule-Walker方法估计信号功率谱，它用到了Yule-Walker AR Estimator块
Magnitude FFT	利用直接法（周期图法）估计信号功率谱
Short-Time FFT	利用短时傅立叶变换估计信号功率谱

【例11.4】 让一段零均值功率为的白噪声通过一AR模型：

$$H(z)=\frac{1}{1-0.1z^{-1}+0.09z^{-2}+0.648z^{-3}}$$

得到输出 $y(n)$ ，再加上一个频率为350 Hz的实正弦信号 $x(n)$ ，然后利用Power Spectrum Estimation库中的各种方法估计上述信号的功率谱。

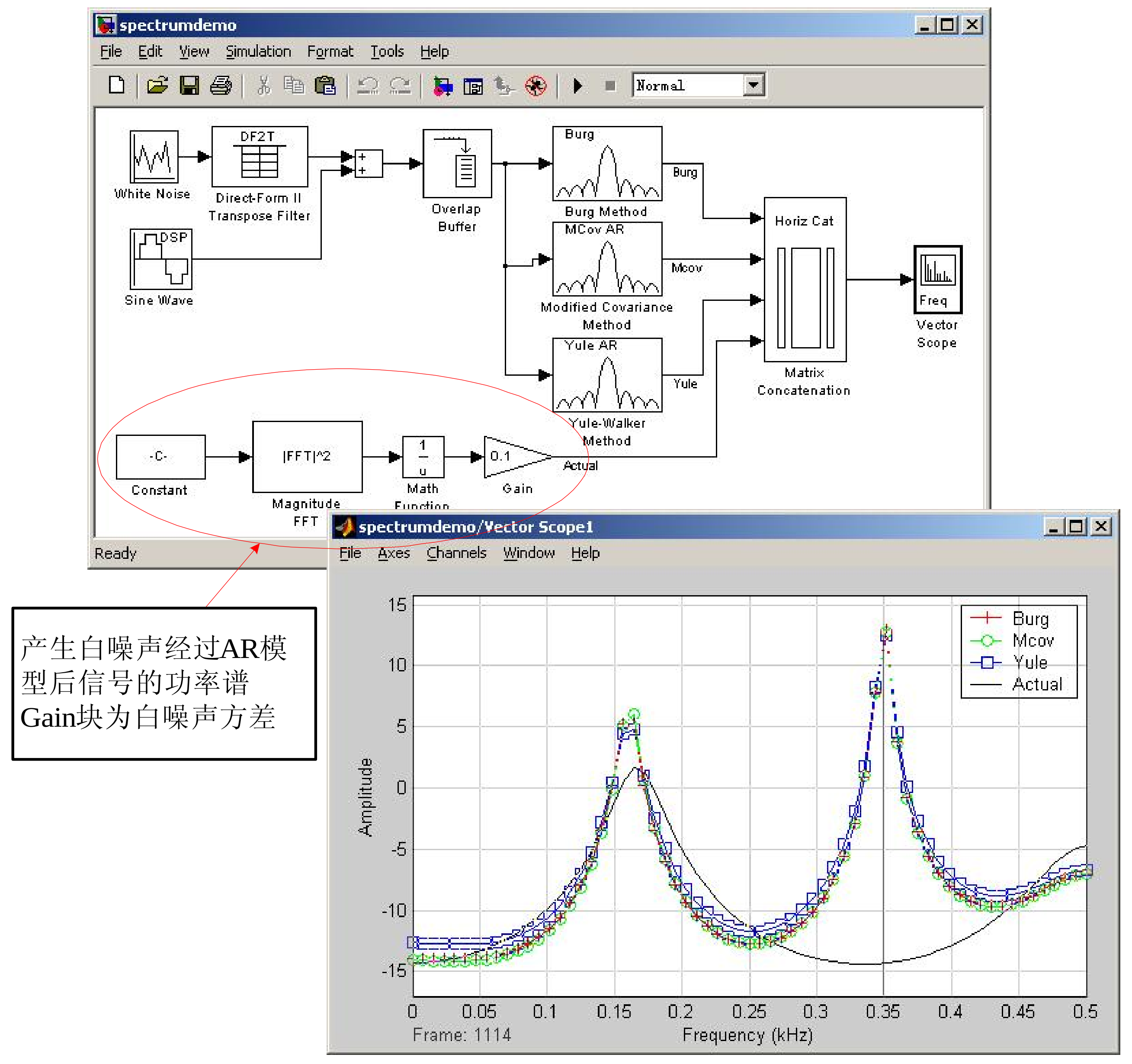

spectrumdemo
File Edit View Simulation Format Tools Help
Normal
White Noise
DF2T
Direct-Form II
Transpose Filter
Sine Wave
Overlap
Buffer
Burg
Burg Method
MCov AR
Modified Covariance
Method
Yule AR
Yule-Walker
Method
Mcov
Yule
Horiz Cat
Matrix
Concatenation
Freq
Vector
Scope
-C-
Constant
|FFT|^2
Magnitude
FFT
Math
Function
0.1
Gain
Actual
Ready
spectrumdemo/Vector Scope1
File Axes Channels Window Help
Amplitude
Frequency (kHz)
Frame: 1114
Burg
Mcov
Yule
Actual
产生白噪声经过AR模型后信号的功率谱
Gain块为白噪声方差

图11.14 功率谱估计

11.2.5 统计

Statistics 库提供了一些常用统计操作模块，如求最小值、最大值、平均值、方差等。大部分模块都支持两种模式：基本模式和运行期模式。基本模式的统计对象是当前仿真周期内的输入，与之前的输入无关，这个输入可以是基于采样的也可以是基于帧的，当然对输入还涉及到过去的输入。

表11.6 Statistics 库

模块名称	模块功能描述
Maximum	求最大值，有基本模式和运行期模式两种模式
Minimum	求最小值，有基本模式和运行期模式两种模式
Median	求中间值，有基本模式和运行期模式两种模式
Sort	排序，使用快速排序算法, 只有基本模式
Mean	求均值，有基本模式和运行期模式两种模式
Variance	求方差，有基本模式和运行期模式两种模式

Standard Deviation	求标准方差，有基本模式和运行期模式两种模式
RMS	求均方根，有基本模式和运行期模式两种模式
Correlation	求相关, 只有基本模式
Autocorrelation	求自相关, 只有基本模式
Histogram	统计直方图，有基本模式和运行期模式两种模式
Detrend	去除信号中的趋势项

【例11.5】 统计一个高斯分布的随机信号的方差、均值并绘制其直方图。

解：Simulink框图如图11.15所示。所需的各个模块及其参数设置如下：

(1)Random Source模块：Source type设置为Gaussian，Samples per frame设置为100。

(2) Histgram模块：Minimum value of input设置为-10，Maximum value of input设置为10，bin设置为21。

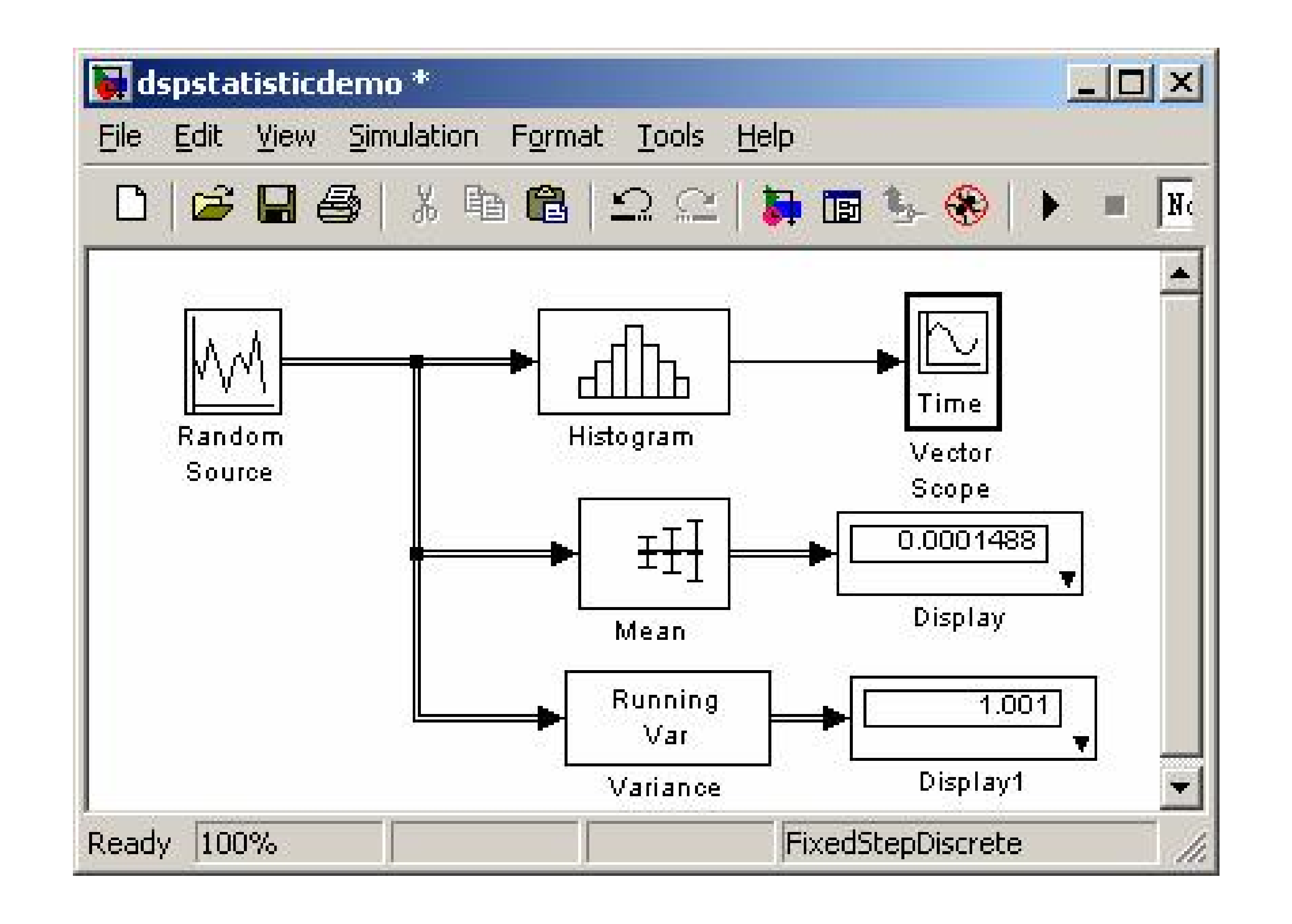

dspstatisticdemo *
File Edit View Simulation Format Tools Help
Random
Source
Histogram
Time
Vector
Scope
Mean
0.0001488
Display
Running
Var
Variance
1.001
Display1
Ready
100%
FixedStepDiscrete

图11.15 信号统计

11.2.6 矩阵操作与线性方程求解

数字信号处理中经常会遇到多通道信号，DSP Blockset 包含了一个 \Math Functions\Matrices and LinearAlegbra库实现对多通道信号（矩阵信号）的处理。常用的操作包括对行或列求和或求积、矩阵乘积和对角线提取等，这些操作包含在Matrix Operations库中， 前面介绍的 Matrix Concatenation模块也在这个库中。另外， Matrix Factorizations 库实现了许多MATLAB 矩阵分解算法供仿真使用；Matrix Inverses库封装了几种求矩阵逆的算法；Linear System Solvers库包含求解AX=B类型的线性方程组的若干方法。表11.7～11.9列出了包含的模块及其功能描述。

表11.7 Matrix Factorizations 库

模 块 名 称	算 法 描 述
LU Factorization	LU矩阵分解
Cholesky Factorization	Cholesky矩阵分解
LDL Factorization	LDL 矩阵分解
QR Factorization	QR矩阵分解
Singular Value Decomposition	SVD矩阵分解

表11.8 Matrix Inverses 库

模 块 名 称	算 法 描 述
LU Inverse	利用LU分解算法求矩阵的逆
Cholesky Inverse	利用Cholesky分解算法求矩阵的逆
LDL Inverse	利用LDL分解算法求矩阵的逆
Pseudoinverse	利用SVD分解算法求矩阵的伪逆

表11.9 Linear System Solvers库

模 块 名 称	算 法 描 述
LU Solver	利用LU分解算法解线性方程组，A阵必须为方阵，B必须与A有相同的行数
SVD Solver	利用SVD分解算法解线性方程组，如果A不是方阵，则所得解为最小二乘解
QR Solver	利用QR分解算法解线性方程组，如果A不是方阵，则所得解为最小二乘解
Levinson-Durbin	利用Levinson递推算法求解Toeplitz型方程组
LDL Solver	利用LDL分解算法解线性方程组，A为对称正定的方阵，B与A有相同的行数
Cholesky Solver	用Cholesky分解法求（即平方根法）解线性方程组，A为对称正定的方阵，B与A有相同的行数
Forward Substitution	前向替代法解线性方程组，其中A是下三角方阵
Backward Substitution	后向替代法解线性方程组，其中A是上三角方阵

【例11.6】 利用LU分解求解线性方程组AX=B的解。其中A＝[1 -2 3; 4 0 6 ; 2 1 5], B=[1 –0.225 –1; 1 2 3]。

解： (1) 利用DSP Constant 模块输入A阵和B阵 。

(2) 在Linear System Solvers库中选择LU Solver模块，它使用LU分解算法求解方程组的解。

(3) 加入display模块用于显示方程组的解。

按照图11.16所示，连好连线，并仿真得到上述方程的解，并在display模块中显示出来。

注意由于B是3×2矩阵，所以求得两组解（两个列向量）。我们来验证一下解的正确性，将解代入AX，这里用一个Matrix Multiply来实现矩阵的乘法，求得的结果显示在display1中，正是B阵。注意这些运算只需一个仿真步长就可以完成，所以仿真时间在这里就没有意义了，因为图10.16本身所表示的框图是一个静态系统，在数学上是一组代数方程。

二者相同

matrixsolver

File Edit View Simulation Format Tools Help

Normal

A

B

A*B

AX=B

1	1
-0.225	2
-1	3

Display1

[1 -2 3;4 0 6;2 1 5]

A阵

[1 -0.225 -1;1 2 3]'

B阵

AX=B (LU)

A

B

[m x m]

X

LU Solver

0.1398	-0.7273
-0.6261	0.3636
-0.1307	0.8182

Display

Ready | 100% | FixedStepDiscrete

